第 3 章

Python 操作数据库

数据库作为重要的基础软件之一，在现代软件开发中有着广泛的应用，特别是对于数字资产的存储、分发、管理，有着非常重要的作用。数据库是指长期存储在计算机内的、有组织的、可共享的数据的集合。通俗地讲，数据库就是专门存储并管理数据的地方。

数据库分为关系数据库与非关系数据库。关系数据库是采用关系模型来组织数据的数据库，即从一个数据可以关联并查询到其他相关联的很多数据。关系数据库的特点是对数据的一致性要求比较高，也就是对事务的支持。关系数据库多用于逻辑关联性较高的场景中，例如各类管理系统、电子商务平台、网上银行、网络支付等应用。其典型的代表是 PostgreSQL 数据库、MySQL 数据库、Oracle 数据库、MS SQL Server 等。关系数据库因为数据的相关性，以及结构相对固定，所以关系数据库在面对大数据的情况下性能会有所欠缺。

非关系数据库又称为 NoSQL，最常见的解释是 non-relational，即非关系，Not Only SQL 也被很多人接受。非关系数据库的特点是易扩展、高性能。数据之间没有关系，所以就非常容易扩展。无形间在架构层面上带来了较强的扩展能力。非关系数据库因为数据结构的简单及数据间的无关性，使非关系数据库具有非常高的读写性，尤其是在大数据量的情况下，表现十分优异。非关系数据库多用于低延时、高并发的场景中，例如股票交易、网络游戏、大数据分析、实时通信等应用。其典型的代表是 MongoDB、Redis、Hadoop HBase 等。非关系数据库不支持 SQL，且不同的数据库语法结构不一致，学习成本较高，另外大多数非关系数据库不支持事务，对数据一致性无法保障，有部分非关系数据也对事务提供了支持，但是与关系数据一样，使用事务时也会降低效率，这样非关系数据库的优势就无法体现出来了。

本章将向读者介绍应用非常广泛的关系数据库 MySQL 与非关系数据库 MongoDB、Redis，并且使用 Python 作为主要编程语言对这几种数据进行操作。

3.1 MySQL 简介及安装

3.1.1 MySQL 简介

MySQL 是典型的关系数据库，其应用十分广泛。MySQL 最早由瑞典的 MySQL AB 公司开发，后被甲骨文公司收购，现在是甲骨文公司旗下的产品。MySQL 是开放源代码的数据库，它采取了双授权政策，分为社区版和商业版。其中社区版本是完全免费的。

MySQL 数据库由于其体积小、速度快、使用成本低，尤其是开放源代码这一特点，很多中小型的网站或者系统会选择 MySQL 作为其数据库。

作为关系数据库，MySQL 将不同的数据保存在不同的表中，而不是将所有数据放在一个大的仓库内，这样就提高了速度和灵活性。

3.1.2 MySQL 特性

1. 运行速度快

MySQL 核心线程是多线程，支持多处理器，从而使处理效率更高。

2. 支持多种存储引擎

MySQL 支持的引擎多达十几种，不同的存储引擎有各自的特点，MyISAM、MEMORY 与 InnoDB 是比较常见的存储引擎。InnoDB 引擎支持事务的处理。

3. 学习成本低

MySQL 使用 SQL 作为数据库的设计语言。SQL 是一种数据库查询和程序设计语言，比较简单，但功能强大。通过简单的语句就可以完成复杂的查询工作。入门比较容易，学习成本低。

4. 存储容量大

在 MyISAM 存储引擎下，MySQL 的存储容量主要受限于操作系统，即操作系统对文件大小的限制，与 MySQL 本身无关。其他引擎的存储量也十分巨大，对于中小系统来讲，完全不必担心存储的容量限制。

5. 安全性高

灵活的授权形式，既可以按照数据库进行授权，又可以按照用户进行授权，连接服务器时，所有的密码传输均采用加密形式，从而保证了密码的安全。

6. 跨平台

MySQL 数据库是跨平台的数据库，可以很好地运行在不同的操作系统上，例如 Linux、Windows 等操作系统。

7. 支持多语言

支持多种开发语言调用，例如 PHP、Python、Java、C、C++、Perl 等。

8. 成本低

MySQL 社区版可以免费获取并使用，MySQL 是一款比较成熟的产品，社区版与商业版在性能方面相差不大，对于大多数中小企业，社区版已经足够使用。

3.1.3 MySQL 安装

MySQL 的官方网站为 https://www.mysql.com/。将该网址输入浏览器中并按回车键，就可以访问 MySQL 的官方网站了，因为该网站存放的服务器在国外，所以从国内访问会较慢，需要耐心等待一下。

成功打开 MySQL 官网后，单击 DOWNLOADS 栏目，即可显示 MySQL 的下载页面，如图 3-1 所示。打开 DOWNLOADS 页面，滑动到页面的最下方，单击 MySQL Community (GPL) Downloads 链接，进入 MySQL 社区版的下载页面，如图 3-2 所示。

因为需要在 Windows 操作系统上安装 MySQL，所以在 MySQL 社区版下载页面内选

择 MySQL Installer for Windows 链接，就可以进入 Windows 安装包的下载界面，如图 3-3 所示。在 Windows 安装包的下载界面有两个下载选项，分别为 mysql-installer-web-community-8.0.23.0.msi 与 mysql-installer-community-8.0.23.0.msi，如图 3-4 所示。

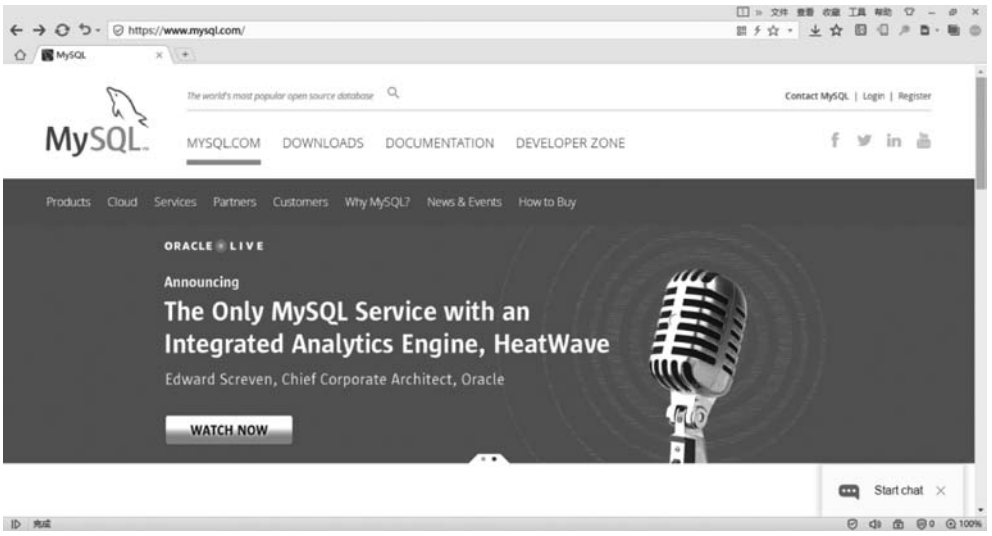

图 3-1 MySQL 官方网站

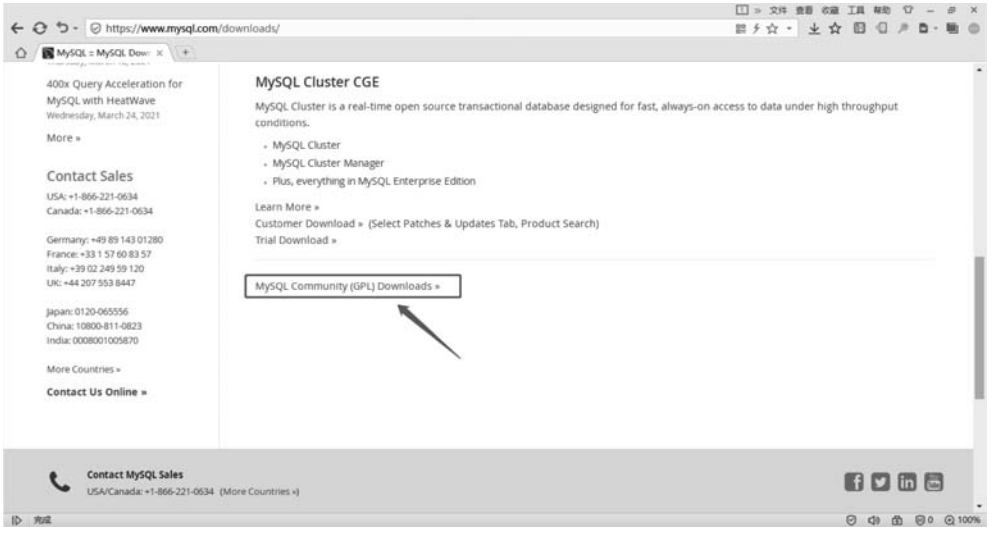

图 3-2 MySQL Community(GPL)DownLoads 链接

其中 mysql-installer-web-community-8.0.23.0.msi 表示通过 MySQL 官方提供的在线安装器进行边下载边安装，而 mysql-installer-community-8.0.23.0.msi 则是完整安装包，一次性下载到本地计算机上后再进行安装，此处选择 mysql-installer-community-8.0.23.0.msi 进行安装，也就是将完整安装包下载到本地后进行安装。

细心的读者可能会发现，MySQL 提供的 Windows 安装包都是基于 x86 架构 32 位操作系统的，如果计算机是 64 位操作系统应该如何选择呢？这里不论是 32 位操作系统还是 64

位操作系统，只能选择 32 位的安装包进行安装。这里的 32 位安装包只是一个打包器，打包器已经将 64 位的 MySQL 打包进去了，在安装 MySQL 的过程中可以选择 64 位的 MySQL，并且 64 位是向下兼容 32 位的，所以选择 32 位安装包将 MySQL 打包进去，既可以适配 32 位的操作系统，也可以兼容 64 位操作系统。

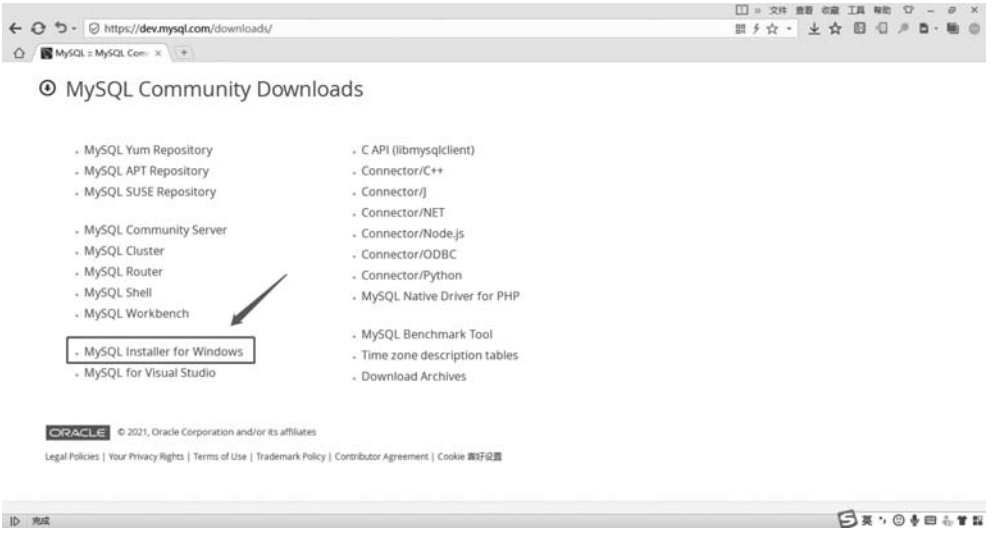

图 3-3　Windows 安装包

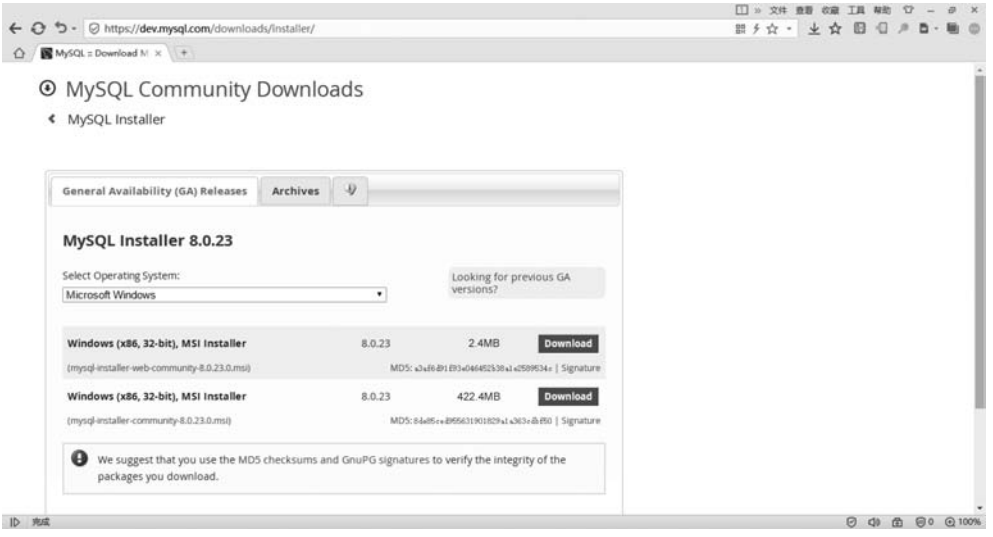

图 3-4　Windows 安装包下载页面

在安装包的下载界面可以看到，当前 MySQL 的最新版本为 MySQL 8.0.x(x 代表数字，例如 MySQL 8.0.23)，在下载界面的 Archives 栏目中的 Product Version 列表中可以看到 MySQL 的版本号突然从 5.7.32 跳到 8.0.0，如图 3-5 所示，其实这期间并不是 MySQL 很久没有维护而导致的断代，而是版本命名的规则发生了改变，可以将 5.7.32 理解为 7.0.0，这样比较容易接受。

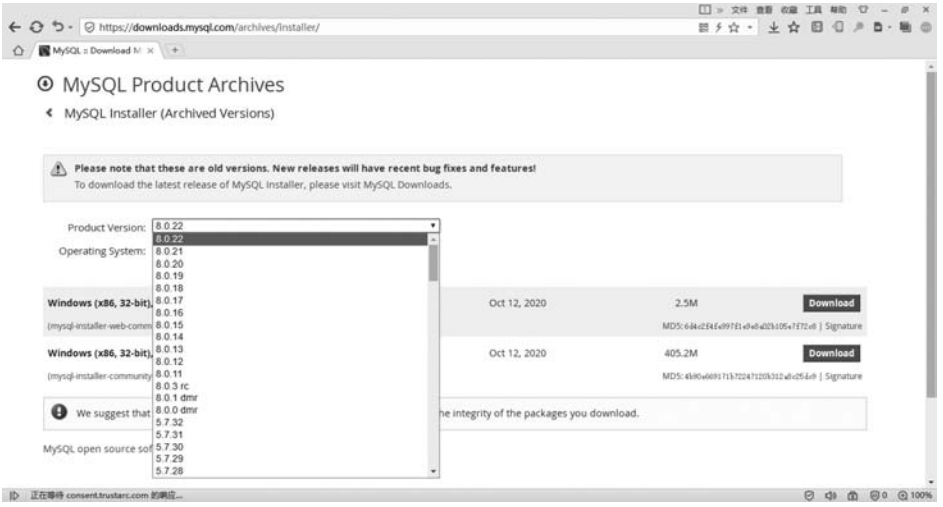

图 3-5　MySQL 历史版本号

双击下载好的 MySQL 安装包(安装包大小为 422.4MB),进入 MySQL 的安装界面,如图 3-6 所示,在安装界面有以下选项:

(1) Developer Default:安装 MySQL 开发所需的所有产品。

(2) Server only:仅安装 MySQL 服务器端。

(3) Client only:仅安装 MySQL 客户端。

(4) Full:安装 MySQL 所有产品和功能。

(5) Custom:用户自定义安装。

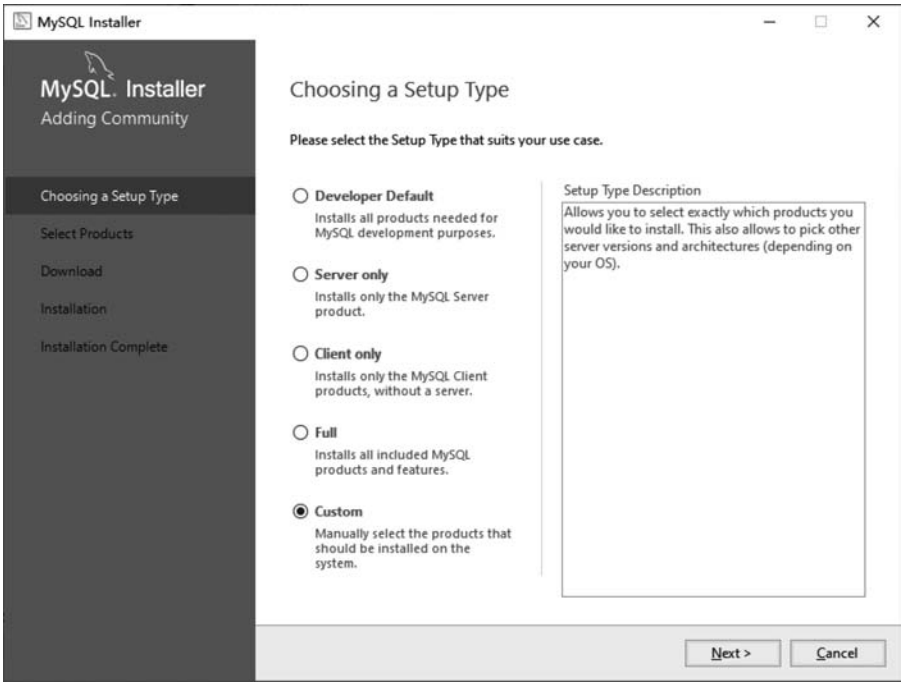

图 3-6　选择 Custom 自定义安装

在此选择 Custom 进行自定义安装，单击 Next 按钮进入自定义安装界面。MySQL 提供的能够安装的产品列表如下。

（1）MySQL Server：MySQL 的服务器端，这是必要的数据库程序，所有的数据库都是在 MySQL Server 下创建的。

（2）MySQL Workbench：MySQL 图形化管理工具。

（3）MySQL Shell：MySQL 命令行管理工具。

（4）MySQL Router：MySQL 的中间件，可以使用 MySQL Router 来做负载均衡。

（5）MySQL Connectors：允许其他语言（例如 Python、C++等）与 MySQL 交互的驱动。

（6）documentation：MySQL 的说明文档。

（7）samples and examples：MySQL 官方提供的案例。

因为本机安装的 Python 版本是 3.9.x 的版本，而 MySQL 官方在本书编写时还没有提供 Python 3.8 以上版本的 Connectors，并且本机也没有安装 Visual Studio，所以在选择自定义安装的时候，需要将这两个选项剔除。

在 Custom 安装界面，提供了两个列表选择框，左侧列表框内是所有可以安装的产品列表，选中左侧列表框中要安装的产品，单击向右的箭头，选中的产品会在右侧列表框中被列出，右侧列表框为即将安装的产品列表。

首先展开 MySQL Server 前方的加号，直到没有加号为止，选中 MySQL Server 8.0.23，然后单击向右的箭头，将 MySQL Server 8.0.23 移动到右侧列表框中。根据上述步骤，继续选择 Applications→MySQL Workbench→MySQL Workbench 8.0→MySQL Workbench 8.0.23，单击向右箭头，将 MySQL Workbench 8.0.23 移动到右侧列表框中；选择 Applications→MySQL Shell→MySQL Shell 8.0→MySQL Shell 8.0.23，单击向右箭头，将 MySQL Shell 8.0.23 移动到右侧列表框中；选择 Applications→MySQL Router→MySQL Router 8.0→MySQL Router 8.0.23，单击向右箭头，将 MySQL Router 8.0.23 移动到右侧列表框中，如图 3-7 所示。

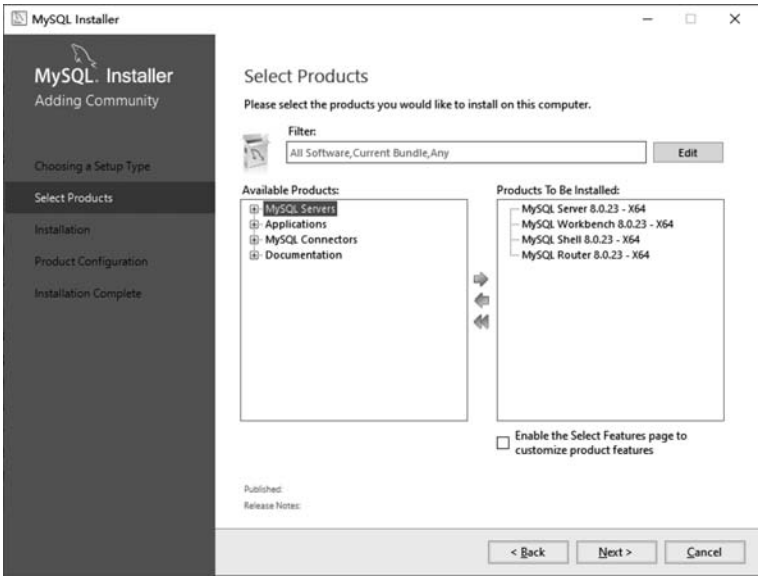

图 3-7　选择需要安装的产品并移动到右侧列表框

单击 Next 按钮后便会显示确认页面，继续单击 Execute 按钮开始安装，在当前页会显示安装进度，需要等待所有的产品出现 Complete，如图 3-8 所示。

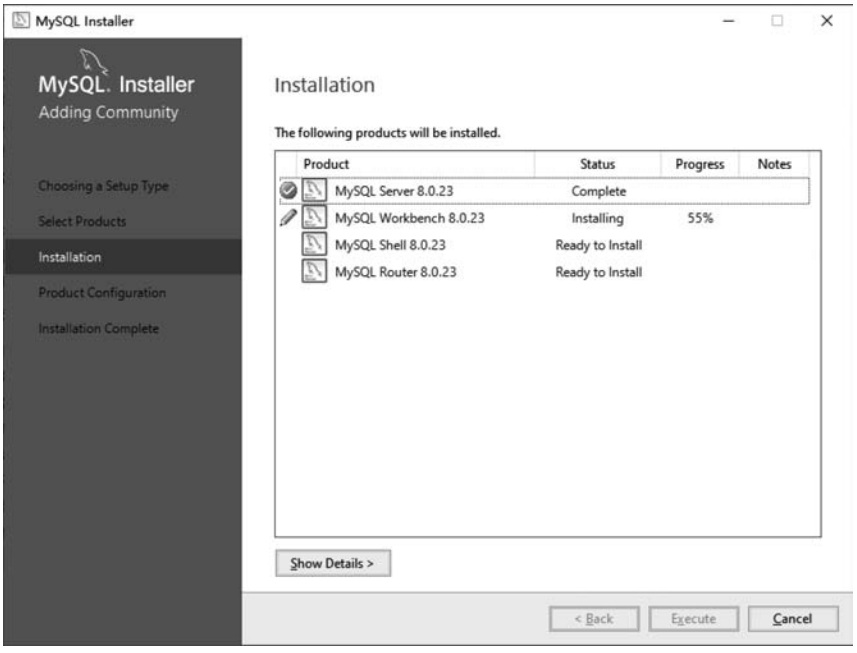

图 3-8　耐心等待所有产品安装完毕

安装完毕后，单击 Next 按钮便会显示确认配置界面，继续单击 Next 按钮进入配置界面，如图 3-9 所示。在配置界面有几个参数需要进行选择与设置。

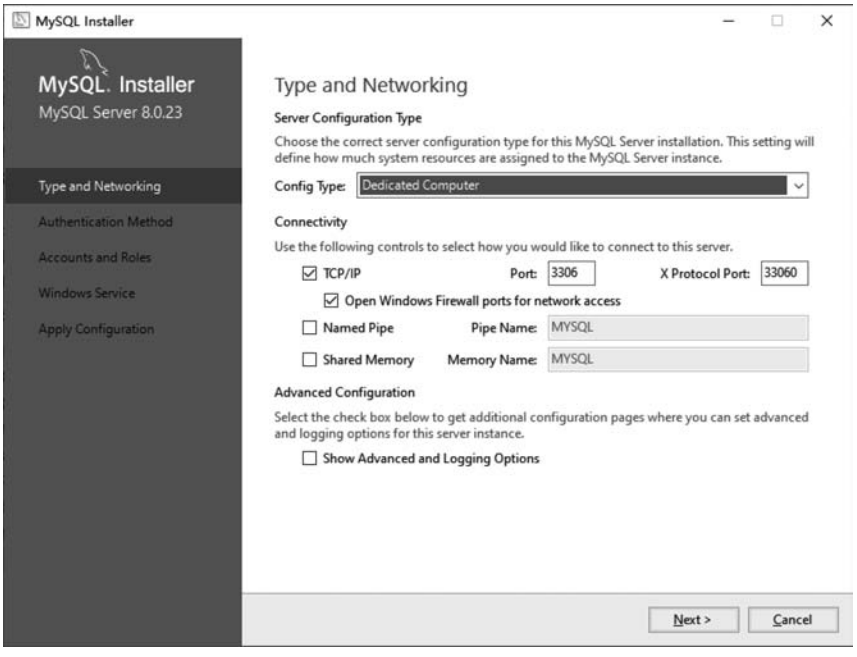

图 3-9　类型及网络设置

Config Type 参数提供了 3 个选项，分别是 Development Computer、Server Computer、Dedicated Computer，这 3 个选项分别对应 MySQL 服务所占用的资源为低、中、高。因为需要在当前机器上开发及运行其他程序，所以选择 Development Computer 即可。

Connectivity 中有以下几个已选中的参数。

（1）TCP/IP：使用 TCP/IP。

（2）Open Windows Firewall ports for network access：允许 Windows 防火墙开放 Port 端口。

（3）Port：TCP/IP 的端口号。

（4）X Protocol Port：MySQL X 协议的端口号。

以上参数保留默认选项即可，单击 Next 按钮，进入认证方式的选择界面，这里有两个认证选项，一个是 Use Strong Password Encryption for Authentication，即使用强力认证。另外一个选项是 Use Legacy Authentication Method，即使用旧的身份验证方法。为了与后面其他程序相兼容，这里选择旧的身份验证方法，如图 3-10 所示，然后单击 Next 按钮。

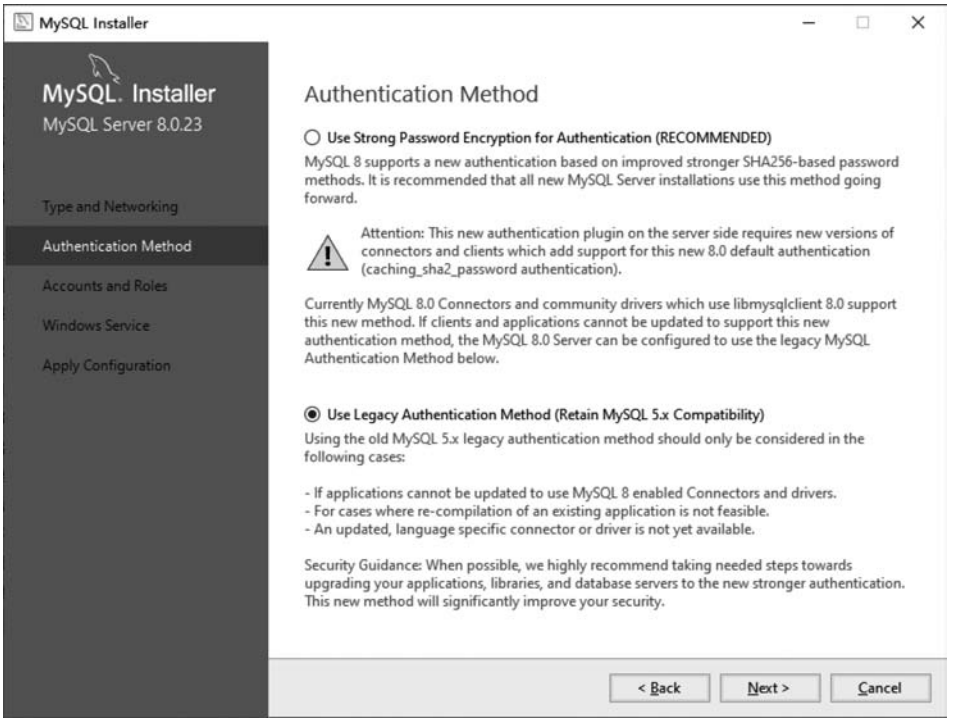

图 3-10　选择认证方式

进入账号设置界面，这里需要设置 root 用户的密码，root 用户是超级用户，拥有对 MySQL 操作的所有权限，除了可以设置 root 密码以外，在当前页面还可以添加新的用户，并且赋予相应的权限。为了操作方便，这里只需设置 root 用户的密码，该密码需要记住，建议设置常用密码，如图 3-11 所示。如果需要对不同的用户区分不同的访问权限，可以通过添加新的用户进行操作。

设置好密码后，单击 Next 按钮进入 Windows Service 设置界面，这里使用默认设置即可，如图 3-12 所示。

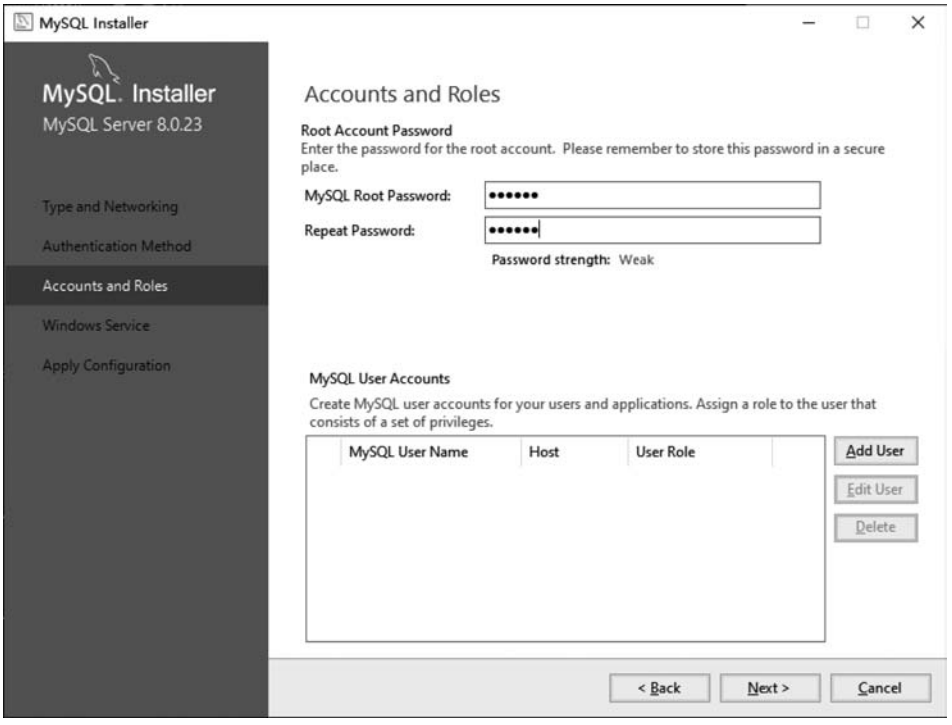

图 3-11　设置用户密码

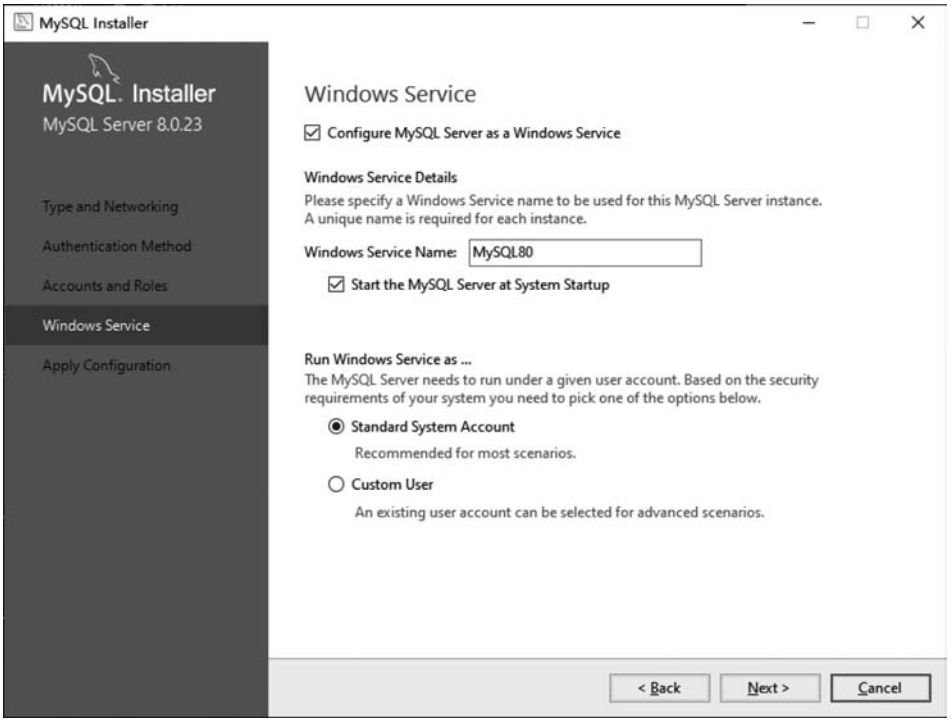

图 3-12　Windows Service 设置界面

单击 Next 按钮进入配置确认界面，如图 3-13 所示，单击 Execute 按钮确认执行相关配置。

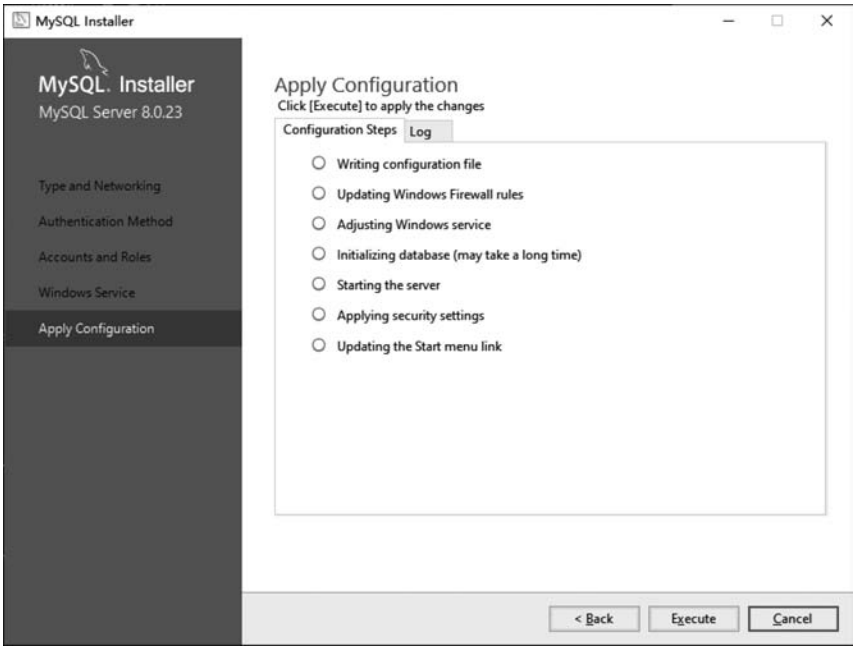

图 3-13　配置确认

单击 Execute 按钮，确认执行后等待配置生效，然后会出现 The configuration for MySQL Server 8.0.23 was successful. Click Finish to continue. 及 Finish 按钮。单击 Finish 按钮完成安装，如图 3-14 所示。

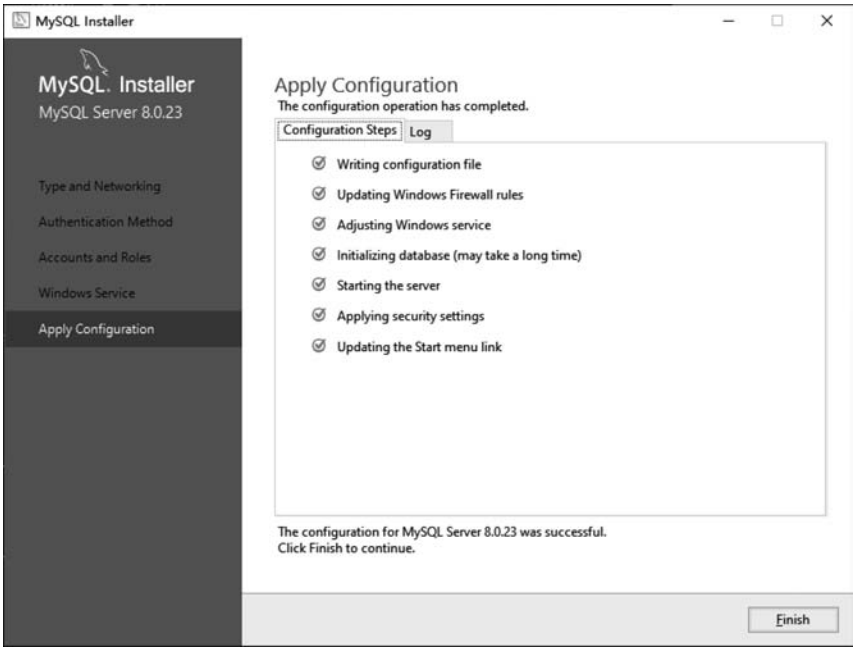

图 3-14　单击 Finish 按钮完成安装

此时完成了 MySQL 的配置，接下来对 MySQL Router 进行配置。单击 Next 按钮进入 MySQL Router 配置界面，该页面选择默认配置即可，单击 Finish 按钮回到产品配置页面，继续单击 Next 按钮进入安装完毕页面，单击 Finish 按钮即完成了 MySQL 的安装及配置，如图 3-15 所示。

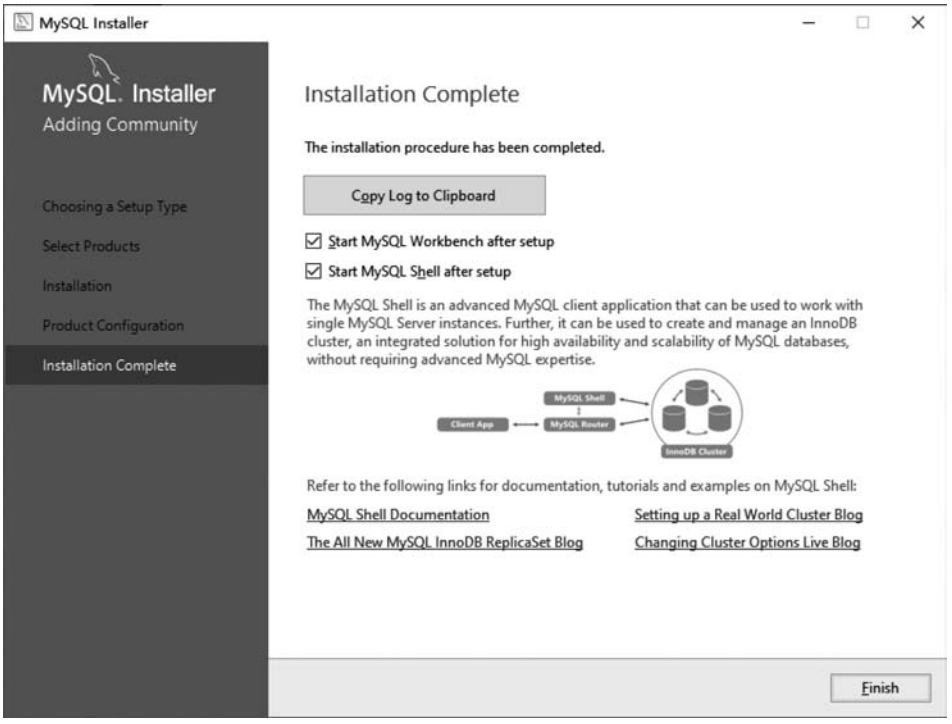

图 3-15 配置确认

因为在安装的最后一步勾选了 Start MySQL Workbench after setup 与 Start MySQL Shell after setup，所以在安装完毕后会弹出两个窗口，分别是 Workbench 与 Shell 窗口。在 Shell 窗口内输入\sql 并按回车键，将交互语言切换为 SQL，前面的\号不能省略，然后输入\connect root@127.0.0.1:3306 并按回车键以便连接数据库，其中 root 是 MySQL 的用户名，127.0.0.1 是本机的 IP 地址，因为 MySQL 在当前机器运行，如果 MySQL 在其他机器运行，则应替换为相应的 IP 地址。3306 是 MySQL 的端口号，此时会提示输入数据库的密码，将安装时设置的 MySQL 密码填入即可。

登录后输入 select version()；就会看到 MySQL 的版本号，如图 3-16 所示。此时 MySQL 已经顺利地安装完毕。

3.1.4 MySQL 可视化工具

前文安装的 MySQL Shell 工具可以通过 SQL 对 MySQL 数据库进行全面管理，包括创建及管理数据库、设计及管理数据表、创建及管理存储过程等，也可以对用户及其权限进行管理，但 Shell 工具界面始终不是太友好，特别对于初学者来讲比较影响使用效率，为此很多第三方产品提供了针对 MySQL 的可视化工具，包括 MySQL 官方也提供了自己的可视化工具。下面将要介绍两个比较常用的 MySQL 可视化工具，一个是由 MySQL 官方提

供的可视化工具 MySQL Workbench，另一个是由第三方提供的可视化工具 Navicat for MySQL。

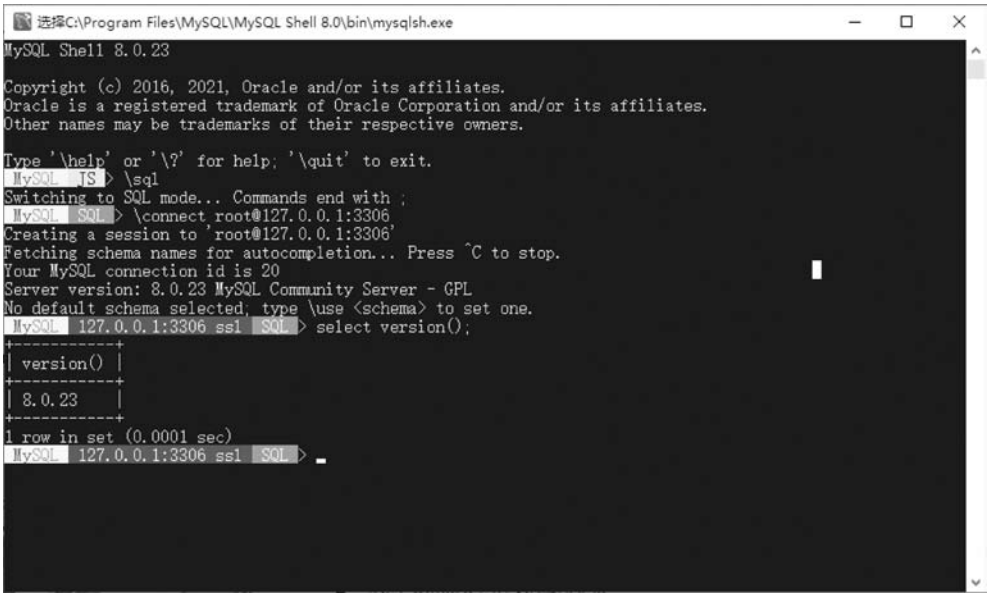

图 3-16　MySQL Shell 操作

1. MySQL Workbench

MySQL Workbench 是 MySQL 官方提供的一款免费的可视化工具，即能在命令行下完成的内容，在可视化工具内都可以完成，反之亦成立。在上文安装 MySQL 的同时，选择并安装了 MySQL Workbench，可以通过“开始”菜单的“最近添加”列表中的 MySQL Workbench 8.0 CE 打开 MySQL Workbench，如图 3-17 所示。

图 3-17　打开 MySQL Workbench 8.0 CE

第一次打开 MySQL Workbench 后会看到欢迎界面，如图 3-18 所示。Workbench 布局采用上、左、右的模式，上部分是菜单项，所有的操作都可以通过菜单完成；左侧有 3 个选项，分别是数据库管理、模型管理及迁移向导；右侧为相对应的操作界面。

在欢迎界面的左下角 Workbench 已经识别出当前机器所安装的 MySQL，可以单击

Local instance MySQL80 卡片来连接本机的 MySQL，也可以通过新建的方式来连接本地的 MySQL。这里选择创建新的连接方式来连接本地的 MySQL，选择菜单 Database→Manage Connections 选项会弹出新建连接的对话框，如图 3-19 所示。

图 3-18 Workbench 欢迎界面

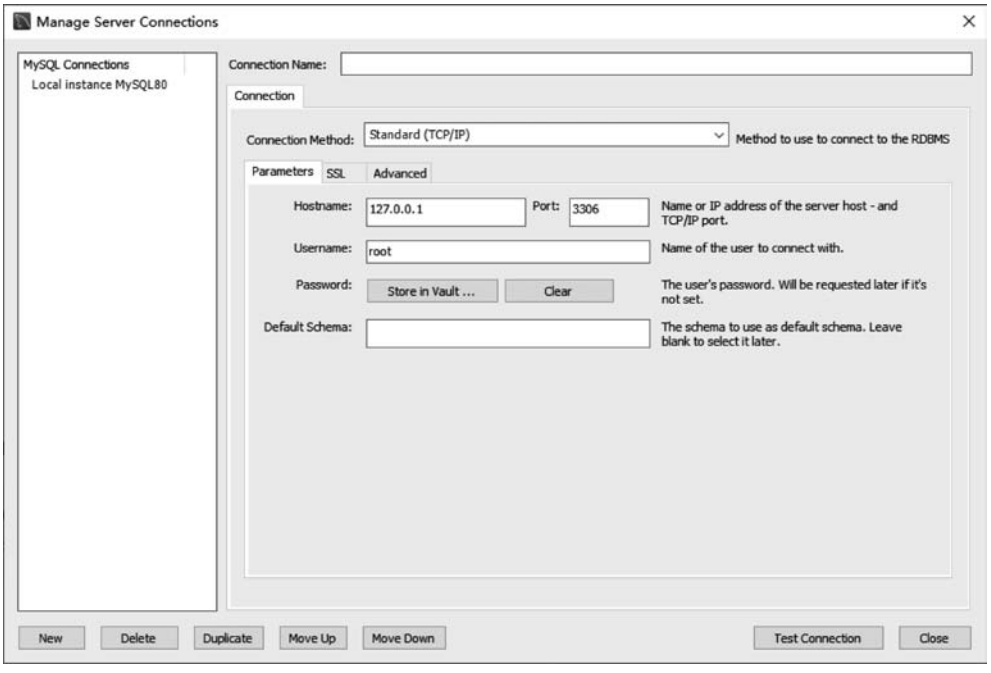

图 3-19 新建 MySQL 连接

在设置参数前，单击左下角的 New 按钮来创建新的连接。新连接有以下参数需要进行设置。

(1) Connection Name：设置连接名称，填写容易辨识的名称，可以使用中文。例如本机 MySQL 数据库。

(2) Connection Method：选择连接数据库的方式，默认选择 TCP/IP。

(3) Hostname：要连接的数据库服务器的 IP 地址，本机将此 IP 设置为 127.0.0.1。

(4) Port：要连接的数据库服务器的端口，默认端口为 3306。

(5) Username：连接 MySQL 的用户名，root 为默认的超级管理员。

(6) Password：连接 MySQL 的密码，通过 Store in Vault 保存。

将以上的参数设置完成后，单击 Test Connection 按钮以便测试是否正常连接，如果参数都正确，则会提示 Successfully made the MySQL connection，如图 3-20 所示。

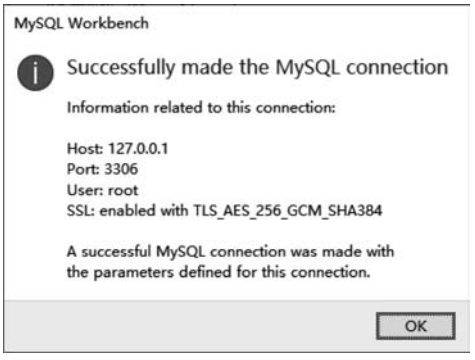

图 3-20　连接 MySQL 成功

新建连接成功后，会在 Workbench 欢迎界面多了一个新建的 MySQL 连接卡片。单击该卡片就可以进入 MySQL 的管理界面了，如图 3-21 所示。

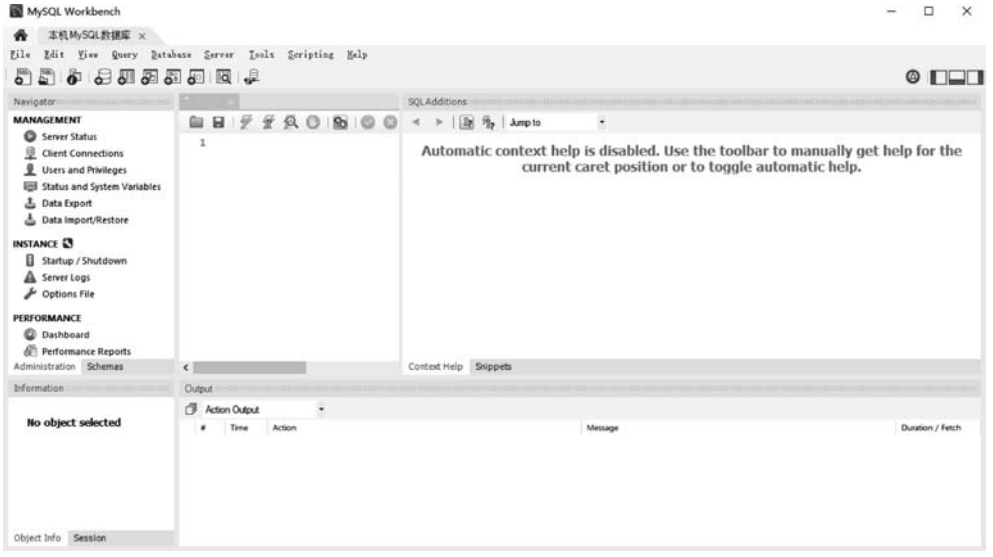

图 3-21　MySQL 管理界面

MySQL 管理界面里面的内容比较多，这里应重点关注对数据库的管理及操作，选择 Navigator 面板下方的 Schemas 选项卡，切换至数据库管理列表。在 SCHEMAS 面板可以

看到默认有一个系统数据库 sys，在 SCHEMAS 面板的空白处右击，在弹出的菜单中选择 Create Schema 选项，在右侧会切换至创建数据库界面，如图 3-22 所示。

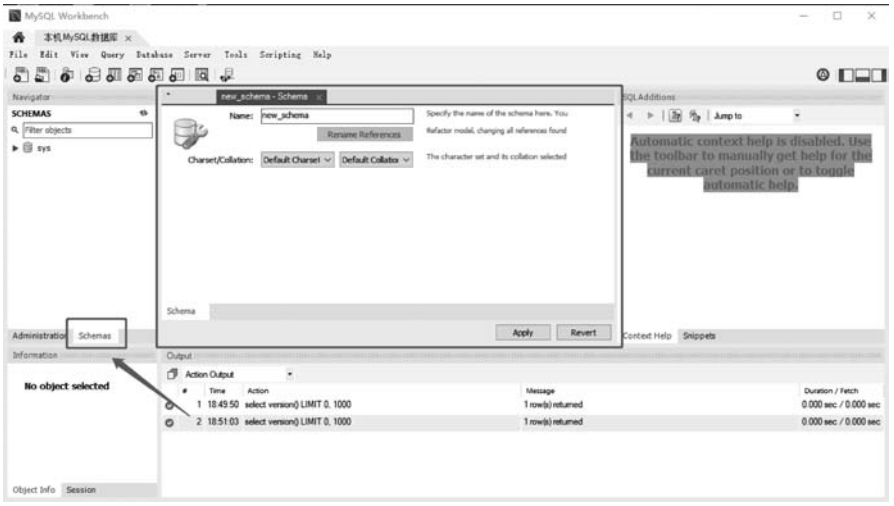

图 3-22　创建新的数据库

创建数据库界面中的参数设置如下。

(1) Name：数据库名，可设置任何名称，但不建议使用中文，例如 mydata。

(2) Charset/Collation：字符编码，建议选择为 utf8 和 utf8_unicode_ci，可以支持中文。

单击 Apply 按钮，弹出一个对话框，提示检查创建数据库的脚本，如图 3-23 所示。在 MySQL 中，一切都可以使用 SQL 进行创建、控制和管理。此时依然单击 Apply 按钮，然后在弹出的对话框中单击 Finish 按钮，完成 mydata 数据库的创建。

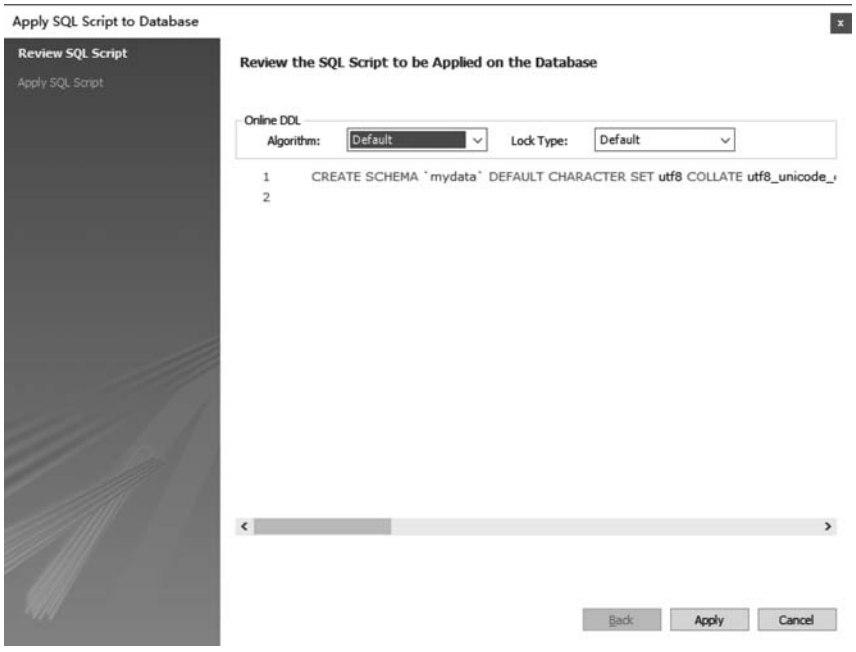

图 3-23　脚本检查对话框

创建完数据库后，在 SCHEMAS 面板中会多出一个刚才创建的 mydata 数据库。单击 mydata 数据库前面的三角按钮便可展开数据库，在数据库下有 Tables、Views、Stored Procedures、Functions，分别为数据表、视图、存储过程、存储函数。这里创建新的数据表，右击 Tables 在弹出的菜单中选择 Create Table 选项，则可进入数据表创建界面，如图 3-24 所示。

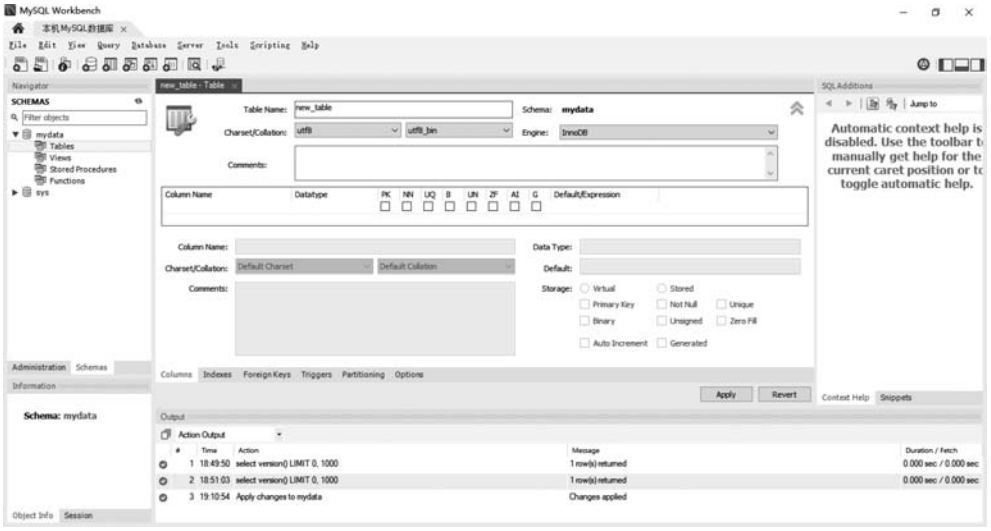

图 3-24　创建数据表

创建数据表界面中的参数设置如下。

(1) Table Name：数据表的名称可以随意填写，例如 mytable，但不建议使用中文。

(2) Charset/Collation：字符编码，这里分别选择 utf8 和 utf8_unicode_ci。

(3) Engine：MySQL 使用的存储引擎，这里选择 InnoDB，InnoDB 用于支持事务。

以上参数设置完成后，需要进一步设计表结构，单击图 3-24 右侧向上的两个箭头，切换到表设计界面，双击 Column Name 下方反选处，即可添加新的字段，如图 3-25 所示。

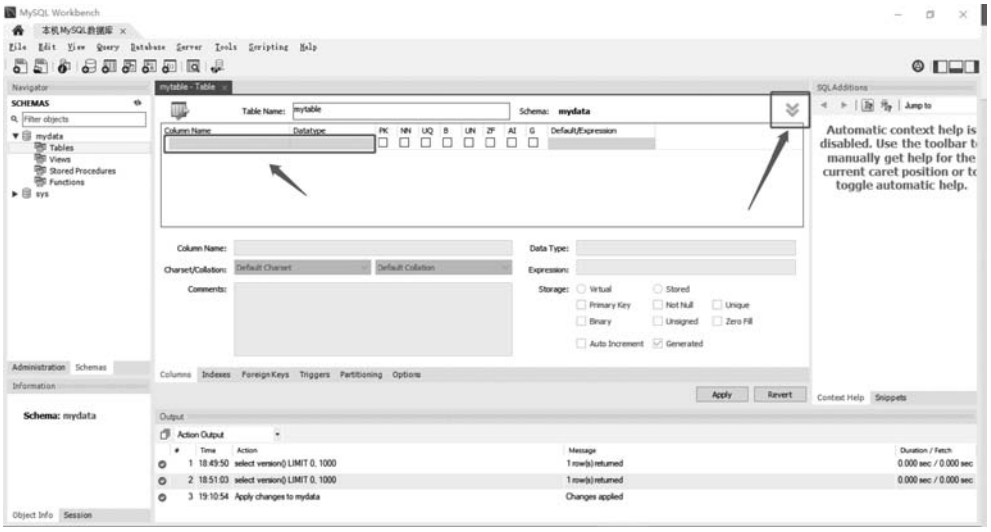

图 3-25　设计字段

在表中添加 3 个 Column Name，分别为 id、username、password，它们的 Datatype 分别为 INT、VARCHAR(45)、VARCHAR(45)，其中 45 表示该字段可以存放最多 45 个字符。对于 VARCHAR 类型的字段需要将下方的 Charset/Collation 参数设置为 utf8、utf8_unicode_ci。对于 id 字段，需勾选 Primary Key、Not Null、Auto Increment 复选框，它们分别代表主键、不为空、自增，如图 3-26 所示。单击 Apply 按钮，同样会弹出脚本检查对话框，继续单击 Apply 按钮及 Finish 按钮完成表设计，右侧 Tables 会出现一个三角形按钮，单击三角形按钮即可看到刚才创建的 mytable 表，右击 mytable 并在弹出的菜单中选择 Select Rows 选项，可以看到表详情，如图 3-27 所示。至此，使用 MySQL Workbench 完成了 MySQL 的连接、数据库的创建及数据表的创建。

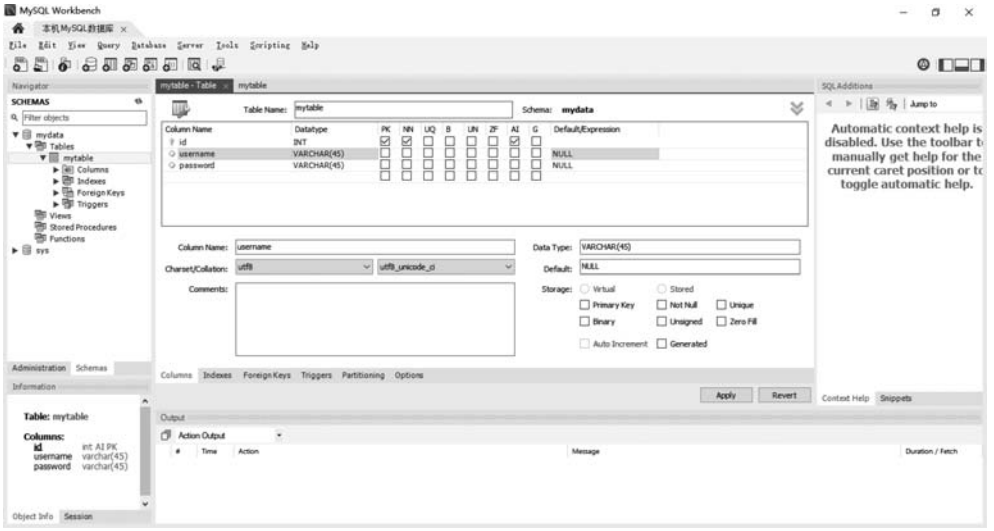

图 3-26　字段设计完毕

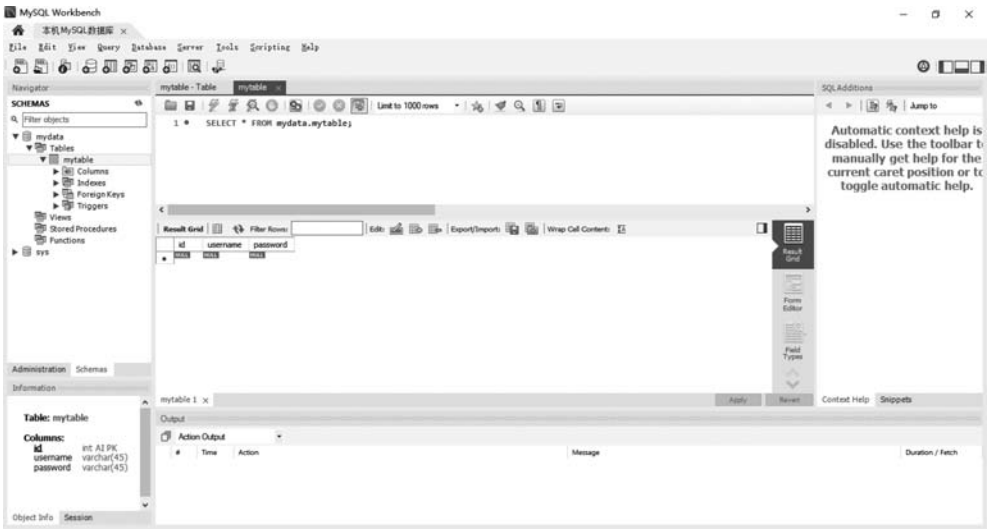

图 3-27　查看表详情

2. Navicat for MySQL

Navicat for MySQL 是一款比较流行的 MySQL 可视化工具，它是一款商业软件，但是因其简洁易用、功能强大的特性，深得从业人员的喜爱。Navicat 是一个系列产品，主要为各种数据库的可视化工具，这里使用的是 Navicat for MySQL。

打开 Navicat 官方网站，网址为 http://www.navicat.com.cn/，选择“产品”栏目，选择 Navicat for MySQL 进行下载并安装，可以免费试用 14 天。其安装比较简单，在此不再赘述。

打开 Navicat for MySQL 界面，如图 3-28 所示，中文界面，界面比较简洁。如果你的界面不是中文，可通过工具→选项→常规→语言选项，选择简体中文，重启 Navicat 后生效。界面默认的布局与 Workbench 的布局一样，也是上、左、右的形式。

界面上部是菜单及常用功能，所有的操作都可以通过此处完成。界面左侧是连接的对象，右侧是对应对象的操作。初次打开左侧列表可能为空白，或者只有一个本地数据库。

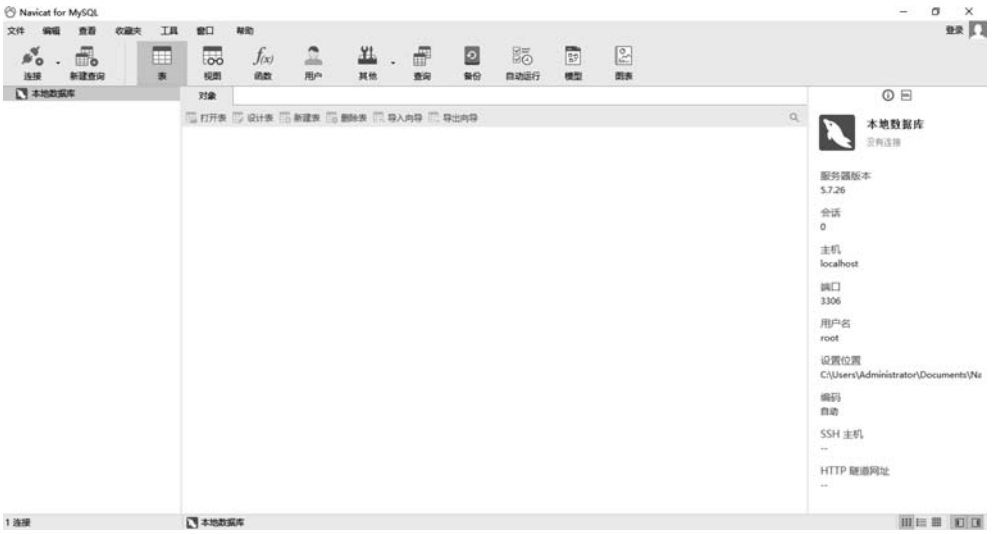

图 3-28　Navicat for MySQL 可视化界面

与 Workbench 一样，可使用 Navicat for MySQL 连接服务器、创建数据库、创建数据表。在创建数据库前，需要先连接目标 MySQL 服务器，单击常用工具中的“连接”按钮，选择 MySQL 选项将会弹出连接对话框，如图 3-29 所示。

该对话框中的参数设置如下。

(1) 连接名：填写容易辨识的名称，可以使用中文，例如本机 MySQL

(2) 主机：填写 IP 地址或者域名，因连接的是本机，所以 IP 地址为 127.0.0.1，亦可以使用 localhost 代替。

(3) 端口：MySQL 安装时设置的默认端口为 3306。

(4) 用户名：登录 MySQL 的用户名，这里使用超级用户 root。

(5) 密码：登录 MySQL 的密码。

以上参数都设置完成以后，单击“测试连接”按钮，如果参数设置正确，则会提示连接成功，然后单击“确定”按钮，在左侧连接列表面板中会出现刚才添加的“本机 MySQL”的连接名。

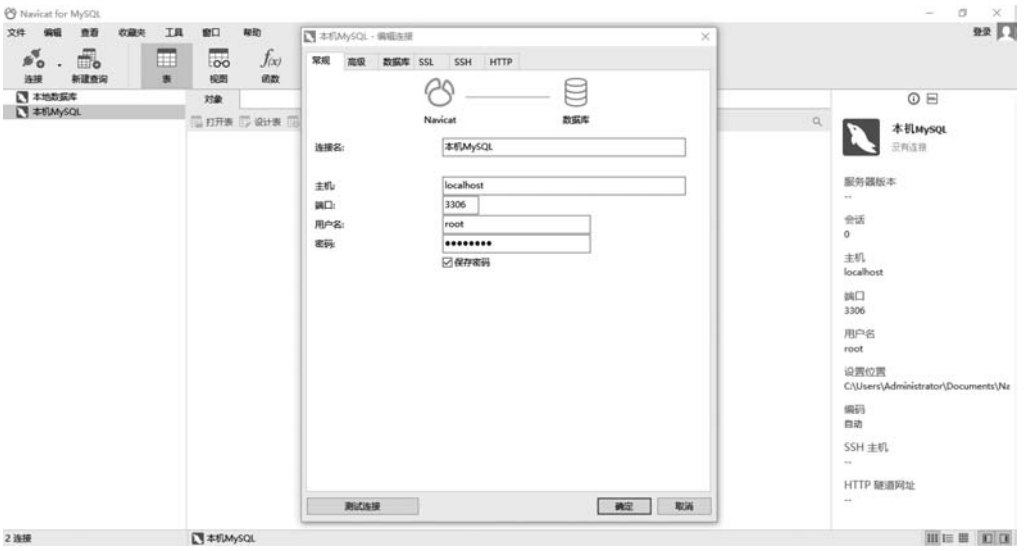

图 3-29　使用 Navicat for MySQL 连接 MySQL 服务器

双击刚才创建的本机 MySQL，将会连接 MySQL 服务器，并且展开该 MySQL 下的所有数据库列表，这时可以看到之前使用 Workbench 所创建的数据库 mydata。双击 mydata 即可看到数据库下的表 mytable，如图 3-30 所示。

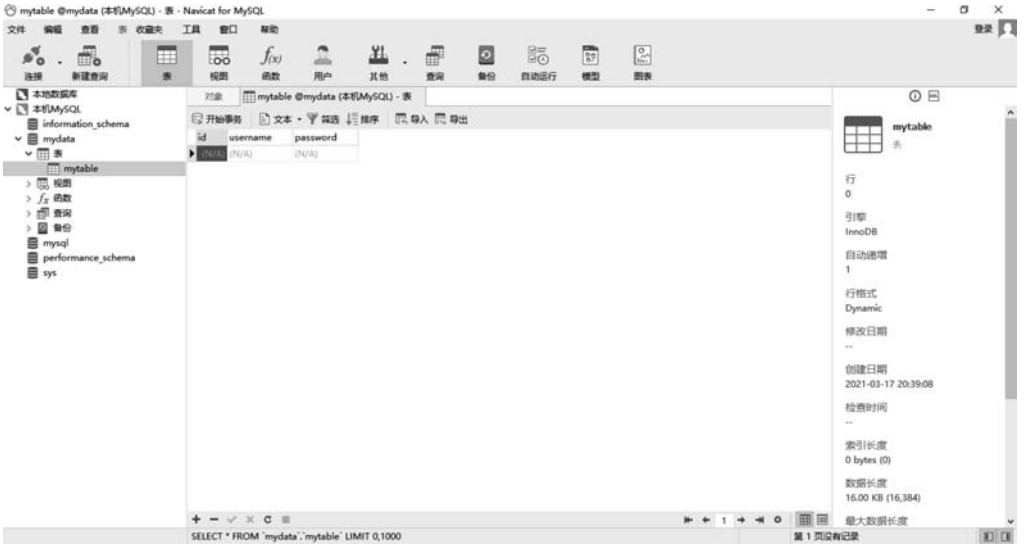

图 3-30　mytable 表

右击连接名“本机 MySQL”，在弹出的菜单中选择“新建数据库”选项来创建一个新的数据库。创建对话框中的参数设置如下。

（1）数据库名：填写具有辨识度的名称，不建议使用中文，例如 newdatabase。

（2）字符集：选择 utf8。

（3）排序规则：选择 uft8_unicode_ci。

单击“确定”按钮则可完成一个名为 newdatabase 数据库的创建，如图 3-31 所示。

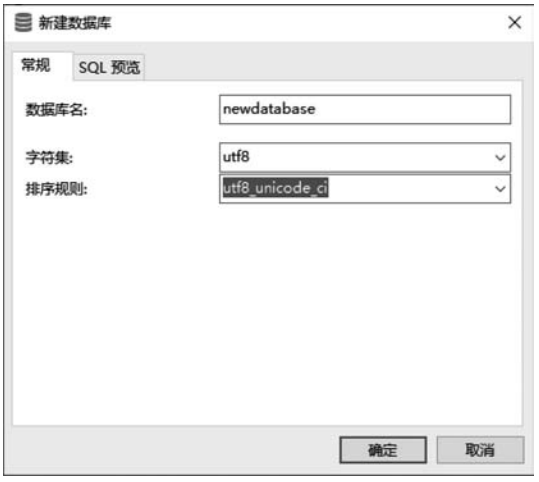

图 3-31　使用 Navicat for MySQL 创建新的数据库

双击左侧数据库列表中已创建的 newdatabase 数据库即打开了该数据库。在其展开的表上右击，在弹出菜单中选择“新建表”菜单，这样就可创建一张新表了，并且进入了表的设计界面，如图 3-32 所示。同样在此表内添加 3 个字段，分别为 id、username、password，字段类型分别为 int、varchar、varchar，varchar 的长度设置为 45，其中 id 将勾选“不是 null”，接着点选“键”及选择“自动递增”。username 与 password 字符集设置为 utf8、排序规则设置为 utf8_unicode_ci。按快捷键 Ctrl+S 保存该表，存储名称为 newtable。

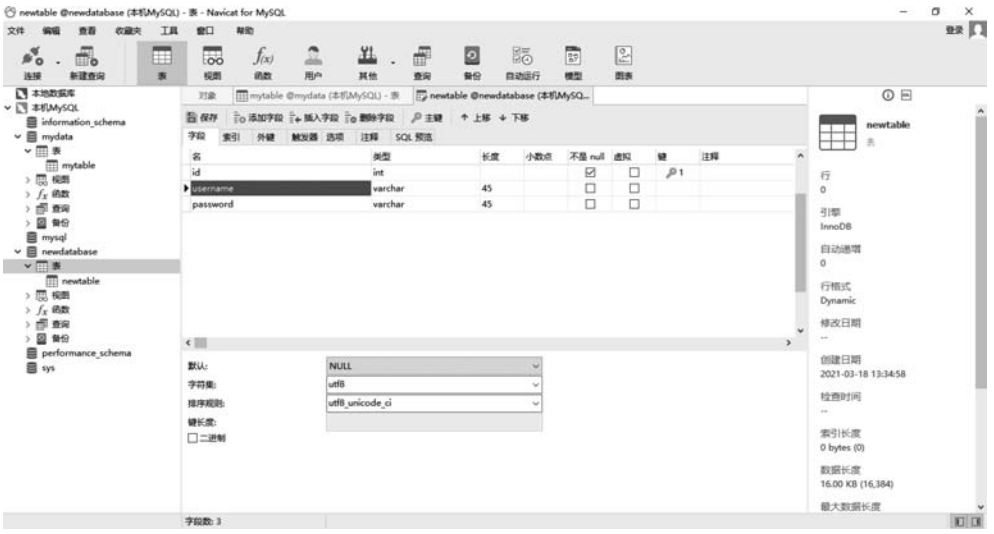

图 3-32　使用 Navicat for MySQL 创建新表

至此使用 Navicat for MySQL 完成了 MySQL 服务器的连接、创建了新的数据库 newdatabase、在 newdatabase 下创建了新的表 newtable。如果要执行 SQL 语句对指定数据库进行操作，则需要先选中该数据库，然后单击常用菜单中的“查询”按钮，单击“新建查

询”按钮即可写入 SQL 语句并执行。例如想要查询 newdatabase 数据库中的 newtable 表中的所有内容，应先单击 newdatabase，然后单击“查询”按钮新建查询，如图 3-33 所示。输入代码如下：

```
select * from newtable
#数据结果为 newtable 下的所有内容
```

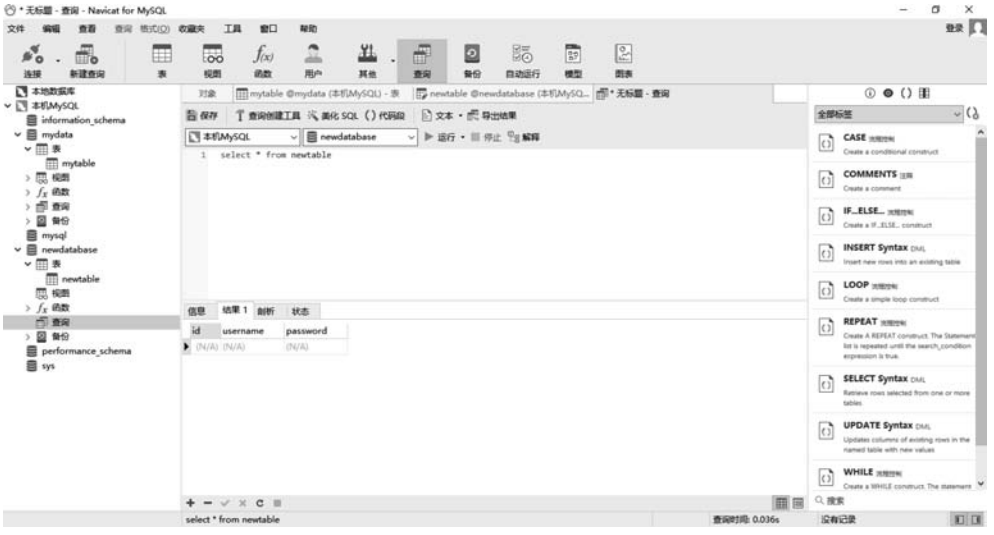

图 3-33　使用 Navicat for MySQL 执行 SQL 语句

3.1.5　MySQL 基础

1. MySQL 数据库与表

在一个 MySQL 服务器中可以同时保存多个数据库，每个数据库可以分别设置不同的访问权限。在使用数据库时通过数据库命名来区分，且命名方式不区分大小写，所以数据库名不能相同。

在一个数据库内可以同时存在多张表，表之间的命名也不能相同。每张表之间应当有相应的字段进行关联，在关系型数据库中数据表一般不应独立存在。

2. 字段的数据类型

MySQL 支持多种数据类型，其类型如表 3-1 所示。

表 3-1　MySQL 数据类型

类　　型	说　　明
bigint	极大整数值，取值范围为－2^63(－9223372036854775808)～2^63－1(9223372036854775807)
binary	固定长度二进制字符串
bit	位字段类型，范围为 1～64
blob	二进制字符串，最大为 65KB(单位为字节)
char	定长字符串，最大长度为 255 字节

续表

类　　型	说　　明
date	日期类型 YYYY-MM-DD
datetime	日期与时间类型 YYYY-MM-DD HH:MM:SS
decimal	精确定点类型,DECIMAL 的数字总长度 M,最大为 65,但是实际能表示的数值范围受精度和标度的限制
double	双精度浮点型
enum	枚举类型
float	单精度浮点型
geometry	任意一种空间类型
geometrycollection	任意一种空间类型集合
int	整数型,字段默认值为 0,取值范围为-2^31(-2147483648)～2^31-1(2147483647)
integer	整数型,字段默认值为 null
json	json 类型
longblob	二进制字符串,最大为 4GB(单位为字节)
longtext	BLOB 或 TEXT 类型
mediumblob	二进制字符串,最大为 16MB(单位为字节)
mediumint	整数型
mediumtext	BLOB 或 TEXT 类型
linestring	线类型,由一系列点连接而成
multilinestring	线集合,包含多条线
multipoint	点集合,包含多个点
multipolygon	多边形集合,包含多个多边形
numeric	精确定点类型
point	空间类型
polygon	多边形类型
real	不精确的双精度浮点型
set	集合类型
smallint	整数型,取值范围为-2^15(-32768)～2^15-1(32767)
text	字符串类型,最大长度为 65535 字节
time	时间类型
timestamp	时间戳类型
tinyblob	二进制字符串,最大长度为 255 字节
tinyint	整数型,取值范围为 0～255
tinytext	字符串类型,最大长度为 255 字节
varbinary	可变长度的二进制数据,取值范围为 1～8000
varchar	字符串类型,最大长度为 65535 字节
year	年份类型

3. 字段的属性

字段拥有以下属性。

(1) Primary Key(主键)：通过唯一索引对给定的一列或多列强制实体完整性的约束。对于每个表只能创建一个 PRIMARY KEY 约束。

(2) Not Null(不为空)：指字段的值不能为空。

(3) Unique(唯一约束)：通过唯一索引为给定的一列或多列提供实体完整性的约束。一个表可以有多个 UNIQUE 约束。

(4) Binary(区分大小写)：字段区分大小写。

(5) Unsigned(无符号)：表示不允许负值。

(6) Zero Fill(填充)：插入数据时，当该字段的值的长度小于定义的长度时，会在该值的前面补上相应的 0。

(7) Auto Increment(自动递增)：当给定某个字段该属性后，该列的数据在没有提供确定数据的时候，系统会根据之前已经存在的数据进行自动增加，以便填充数据。

(8) Generated(虚拟自增列)：该列由其他列计算而得。

4. MySQL 事务

MySQL 事务主要用于处理操作量大且复杂度高的数据。例如订单系统，用户购买了一件商品，下了一个订单并支付，此时要生成一个订单记录，在用户未支付时订单状态为未支付，商品的库存数量也保持不变，当用户支付后，此时要将订单状态修改为已支付，并且同时将商品的库存数量减 1，这些数据库操作就构成了一个事务。

在 MySQL 中，只有 InnoDB 存储引擎才支持事务，事务的特点是要么全部执行成功，要么全部执行失败，不会存在一半成功一半失败的情况。例如上面的例子，使用事务操作时，不会出现订单的支付状态被修改了，而商品的库存数量却没有减少的情况。

在 MySQL 中，使用 BEGIN 开始一个事务，使用 ROLLBACK 对事务进行回滚，使用 COMMIT 对事务进行确认。示例代码如下：

```
BEGIN -- 开始事务
select * from order;
insert into user values("jack","123456");
insert into newtable values("abcdef","python 开发");
COMMIT -- 提交事务
```

3.2 SQL

SQL(Structured Query Language)即结构化查询语言，是一种数据库查询和程序设计语言，用于存取数据库及查询、更新和管理关系数据库系统。

SQL 是高级非过程化编程语言，与普通编程语言不同，它不需要对数据进行定义、存储等。它是基于高层数据结构之上工作的语言。结构化查询语言可以嵌套，这使它具有极大的灵活性和强大的功能。

SQL 常见的操作是对数据表进行创建，以及对数据表中的记录进行读取、增加、编辑、删除等操作，利用 Navicat for MySQL 可以直接对表进行操作，大多数时候使用 SQL 语句对数据表进行操作。

SQL 本身不区分大小写，但是不同的操作系统有可能会区分大小写，例如在 Windows 操作系统中，SQL 不区分大小写，而在 Linux 系统中，有可能会因为大小写而产生一些意料之外的问题，所以在写 SQL 语句的时候，大小写应当保持统一。

1. 使用 SQL 创建表

使用 SQL 语言创建表的语法如下：

```
CREATE TABLE 表名 (
字段 1  字段 1 类型  字段 1 属性,
字段 2  字段 2 类型  字段 2 属性
);
```

打开 Navicat for MySQL，双击指定的数据库，单击快捷菜单中的“查询”→“新建查询”按钮便可创建一个新的查询。在新建查询界面可以输入 SQL 语句并执行，例如创建一个 newtable 数据表并设置 3 个字段，分别为 id、user、pass，对应的数据类型分别为 int、varchar、varchar，创建表的 SQL 代码如下：

```
CREATE TABLE 'newtable' (
    -- 创建字段 id、设置为整型、设置不为空、设置自动增长
  'id' int NOT NULL AUTO_INCREMENT,
    -- 创建字段 user、设置为字符串型，字符宽度为 45、设置为 utf8 格式、设置允许为空、-- 设置
默认为空
  'user' varchar(45) CHARACTER SET utf8 COLLATE utf8_unicode_ci NULL DEFAULT NULL,
    -- 创建字段 pass、设置为字符串型，字符宽度为 45、设置为 utf8 格式、设置允许为空、-- 设置
默认为空
  'pass' varchar(45) CHARACTER SET utf8 COLLATE utf8_unicode_ci NULL DEFAULT NULL,
    -- 将 id 设置为主键
  PRIMARY KEY ('id') USING BTREE
)
```

单击“运行”按钮会弹出信息栏，并在信息栏内输出 OK 及 SQL 代码运行的时长，在左侧刷新后会出现刚才创建的 newtable 表，如图 3-34 所示。如果没有出现 OK 按钮，则说明 SQL 代码存在错误，在错误代码中会提示大概在哪个位置有错误，根据错误提示修改代码直到输出 OK 为止。

在上面的代码中，使用双横线--可以为 SQL 代码添加注释。需要注意的是注释与双横线之间有空格，如果不写空格 SQL 则会报错。

双击左侧新建的表即可进入刚刚创建的表中，该表没有使用可视化界面来创建，而是完全使用 SQL 语言进行创建的。

2. 使用 SQL 读取数据

使用 SQL 语言读取表的语法如下：

```
SELECT 字段名 FROM 表名
```

同样使用 Navicat for MySQL 来执行读取语句，按照前文创建表的方式进入查询界面，查询刚才创建的 newtable 表中的所有字段，则 SQL 代码如下：

图 3-34　Navicat for MySQL 下使用 SQL 创建新表

```
-- 查询 id,user,pass 字段
SELECT id,user,pass FROM newtable
```

因为创建的 newtable 表中没有任何数据，所以将输出一张空表，如图 3-35 所示。

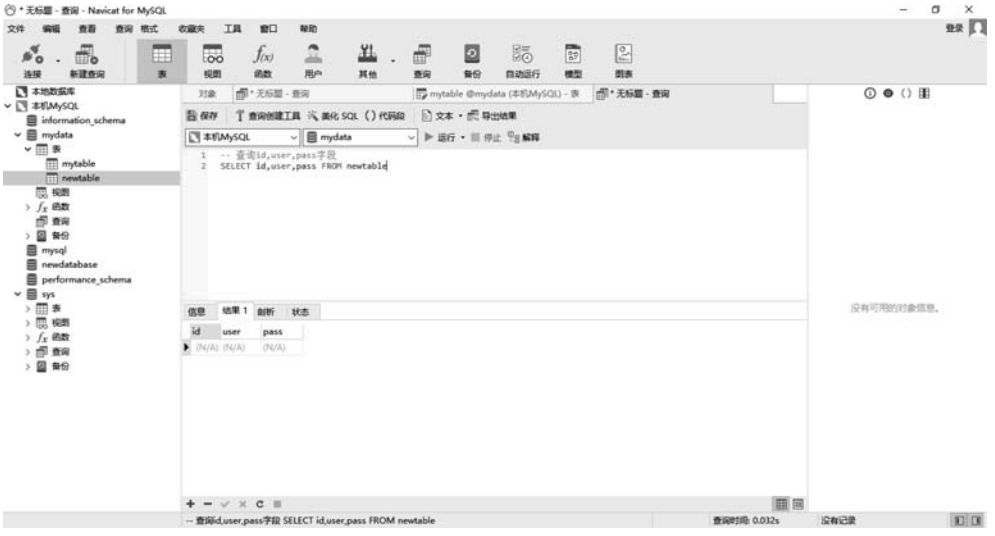

图 3-35　Navicat for MySQL 下使用 SQL 查询表

当表中字段特别多时，如果想要将所有的字段一次性查询出来，则可以使用通配符 * 来代替所有字段进行输出，SQL 代码如下：

```
-- 查询 id,user,pass 字段
SELECT * FROM newtable
```

一般情况下，使用 SQL 查询表时都会输出结果，如果查询语句内有错误，则会输出错误信息，并指明哪里发生了错误，需要进行修改，直到输出结果为止，在其信息面板也会输出 OK。

3. 使用 SQL 增加记录

使用 SQL 语言增加记录的语法如下：

```
INSERT INTO 表名称 (列 1, 列 2,...) VALUES (值 1, 值 2,....)
```

如果按照字段顺序来添加内容并且每个字段都有内容，则(列 1,列 2,...)可以省略。例如在 newtable 中增加一条记录，同样先进入 Navicat for MySQL 查询界面，SQL 代码如下：

```
-- 为 newtable 新增一条记录,id 为自增字段,所以不用设置 id
INSERT INTO newtable (user,pass) VALUES ("jack","123456")
-- 省略列
INSERT INTO newtable VALUES ("jack","123456")

#输出结果为
> Affected rows: 1
> 时间: 0.014s
```

使用 SQL 语言新增记录时，如果新增记录被正常地插入了表中，则在结果中会显示影响了多少行，但插入数据不会显示结果。如果需要查看插入的数据，则可以再次使用 SELECT 查询语句来查询表的内容。例如要查看刚才插入的记录，SQL 代码如下：

```
SELECT * FROM newtable
```

输出结果如图 3-36 所示。

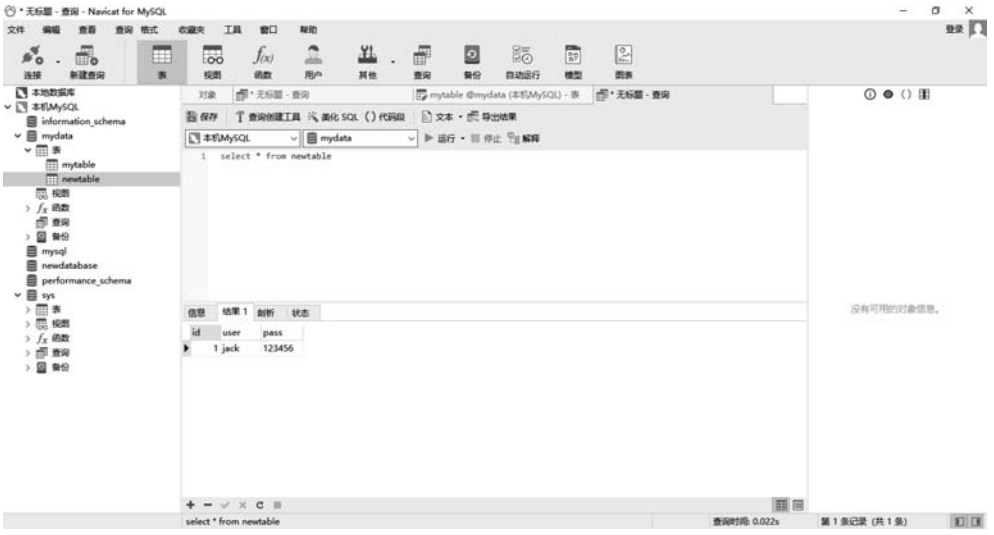

图 3-36 SQL 查询结果

4. 使用 SQL 编辑记录

使用 SQL 语言编辑记录的语法如下：

```
UPDATE 表名称 SET 列名称 = 新值 WHERE 列名称 = 某值
```

更新表中的记录应使用 UPDATE 关键字，例如要将上文查询到的 jack 的密码修改为 abcd，SQL 代码如下：

```
UPDATE newtable SET pass = "abcd" WHERE user = "jack"
输出结果为
> Affected rows: 1
> 时间: 0.013s
```

修改指定的行的记录时需要在后面附上条件语句，例如上面的 WHERE user＝"jack" 就是条件语句，WHERE 是条件语句的关键字，即指定 user 字段中记录为 jack 这一行的记录，并将其对应的 pass 字段修改为 abcd。

与 INSERT 一样 UPDATE 语句执行后并不展示任何结果，只是输出代码运行的结果。如果代码执行顺利并执行完毕，则会返回代码影响的行数，以及代码执行的时间。如需要查看结果，同样可使用 SELECT 将其结果查询出来，如图 3-37 所示。此时 jack 行的 pass 已经由原来的 123456 修改为 abcd 了。

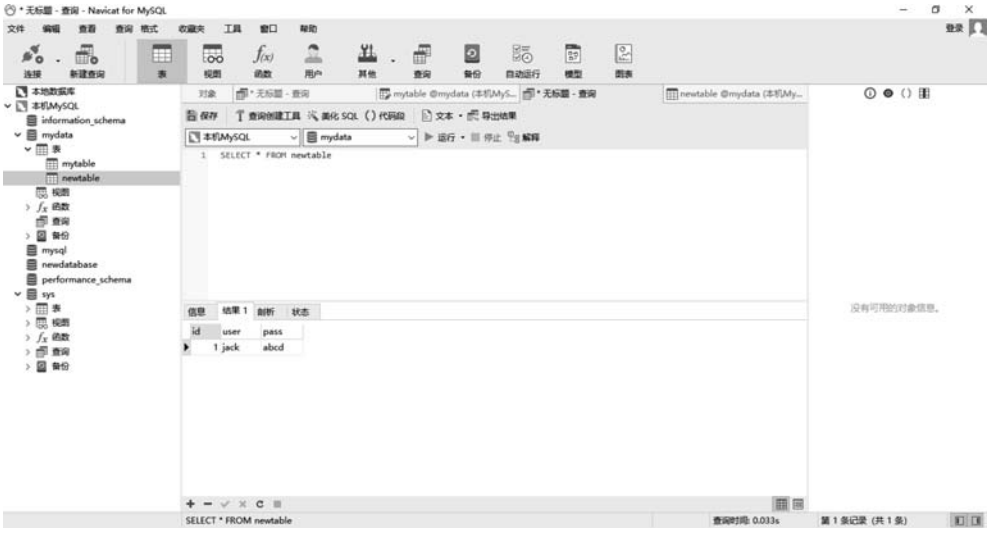

图 3-37 修改字段值后的结果

5. 使用 SQL 删除记录

使用 SQL 语言删除记录的语法如下：

```
DELETE FROM 表名称 WHERE 列名称 = 值
```

与 UPDATE 类似，在删除记录时需要使用条件语句指定要删除哪一行数据。例如要将上文的 jack 记录删除，SQL 代码如下：

```
DELETE FROM newtable WHERE user = "jack"

#输出结果为
> Affected rows: 1
> 时间: 0.016s
```

注意 删除操作如果不指定条件语句，则会将该表中所有的数据删除。

删除操作也只会返回代码执行的状态，而不会返回表中的内容。使用 SELECT 来查询 newtable，会发现 jack 所在的行已经被删除了，如图 3-38 所示。

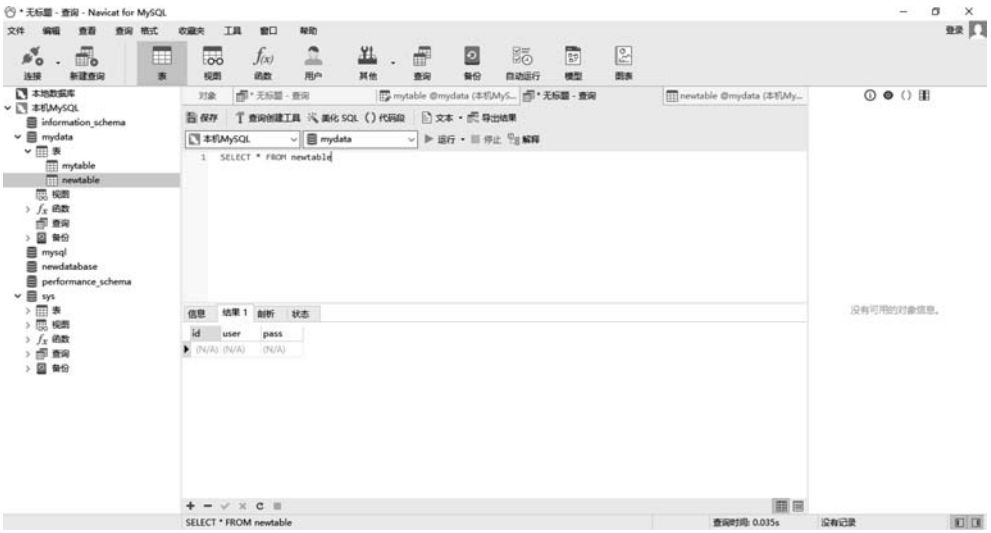

图 3-38 删除记录后的结果

6. SQL 条件语句

前文进行删除、编辑的时候，需要指明条件以便对哪一行进行操作，使用 WHERE 关键字进行指定。特别是删除操作，如果不指定删除某一行，则会将表中的所有数据删除。

在 SQL 语言中使用 WHERE 关键字设置条件。对表的查询（SELECT）、删除（DELETE）、更新（UPDATE）等需要对具体记录进行的操作，都需要使用 WHERE 语句设置操作条件。示例代码如下：

```
-- 查询 user 字段为 jack 的记录
SELECT * FROM newtable WHERE user = "jack"

-- 修改 user 字段为 jack 的记录，将该记录中的 pass 设置为 123
UPDATE newtable SET pass = "123" WHERE user = "jack"

-- 删除 user 字段为 jack 的记录
DELETE FROM newtable WHERE user = "jack"
```

WHERE 后面除了可以使用等号操作符之外，也可以使用很多其他的操作符，具体操

作符列表如表 3-2 所示。

表 3-2 操作符说明

操作符	说 明
=	等于
<>	不等于
! =	不等于
<	小于
>=	大于或等于
<=	小于或等于
BETWEEN	在某个范围内
LIKE	匹配某种模式

在 WHERE 中还可以使用 AND、OR 进行多条件判断，示例代码如下：

```
-- 查询 user 字段为 jack 并且 pass 字段为 123 的记录
SELECT * FROM newtable WHERE user = "jack" AND pass = "123"

-- 查询 user 字段为 jack 或者为 marry 的记录
SELECT * FROM newtable WHERE user = "jack" OR user = "marry"
```

7. SQL 去重

使用 DISTINCT 关键词查询指定字段不重复的记录，语法如下：

```
SELECT DISTINCT 字段 FROM 表
```

当统计 newtable 表中所有的用户时，会发现 user 字段中有两个相同的 jack，如图 3-39 所示。

id	user	pass
4	jack	123
5	marry	456
6	jack	111

图 3-39 表中 user 字段含有相同的值

使用 SELECT 进行查询时，会将两条相同的数据都查出来，使用 DISTINCT 关键字就可以显示去重的记录，代码如下：

```
SELECT DISTINCT user FROM newtable

#输出结果为
user
jack
marry
```

8. 查询排序

SQL 中使用 ORDER BY 语句对结果集进行排序，示例代码如下：

```
SELECT * FROM newtable ORDER BY id
```

此时将会使用 id 字段进行升序排序，如果希望使用降序排序，则可加上 DESC 关键字，

代码如下：

```
SELECT * FROM newtable ORDER BY id DESC
```

在使用排序时，指定的排序字段应可以进行排序，否则可能排出的结果与想要的结果不一致。例如排序的字段类型指定为字符串，且包含的记录中有数字样式的字符串、字母样式的字符串及符号样式的字符串，此种情况的排序结果是不可控的。

9. 返回指定行数

在 MySQL 中使用 LIMIT 关键字获取指定条数的数据，其语法如下：

```
SELECT * FROM 表 LIMIT 起始行数,条数
```

当数据表中的记录特别多的时候，例如当在 newtable 表中有一百万行记录时，此时想要对最早入库的 5 条记录进行操作，就需要使用 LIMIT 来限制读取记录的条数。否则将一百万行记录都读取出来再进行操作，这个时间将会很长，稍差点的计算机可能还会出现卡顿现象。

使用 LIMIT 关键字的示例代码如下：

```
SELECT * FROM newtable LIMIT 0,10
```

注意该段代码的含义，此处并不是读取从 0 条到第 10 条的记录，而是从第 0 条开始，读取 10 条记录，这里面的含义是不一样的。再举个例子，代码如下：

```
SELECT * FROM newtable LIMIT 5,10
```

上面这段代码不是读取从第 5 行到第 10 行的记录，而是从第 5 行起但不包含第 5 行，读取 10 行记录，也就是读取的是第 6 行到第 15 行的记录。这里比较容易混淆，读者需要注意。

3.3 使用 Python 操作 MySQL

3.3.1 MySQL 操作模块

在 MySQL 官方网站中，提供了便于其他语言连接并对 MySQL 进行操作的驱动程序 Connectors，也包括 Python 语言，但是当前 MySQL 官方提供的连接驱动在本书写作期间最高只支持 Python 3.8，而本书使用的 Python 是 3.9.x 版本，所以无法使用 MySQL 官方提供的驱动程序。

除了 MySQL 官方提供的操作模块之外，还有很多非常流行的第三方模块，例如 PyMySQL 模块与 mysqlclient 模块，这两个模块都支持 Python 3，两个模块各有特点。

PyMySQL 模块使用纯 Python 开发，安装与使用都比较简单，相对于 mysqlclient 模块来讲处理速度慢一些，适用于中小型的项目。

mysqlclient 模块是基于 C 语言的，所以其速度比较快，但其缺点是安装起来比

PyMySQL 复杂，对新手来讲不太友好。

两个模块除了安装有所区别以外，对于使用 Python 操作数据库方面大同小异，因此本书将使用 PyMySQL 模块来操作数据库。

3.3.2 使用 Python 操作 MySQL

因 PyMySQL 是第三方模块，所以在使用前需要对其进行安装。首先打开 PyCharm 编辑器，然后单击 View→Tool Windows→Terminal 打开终端窗口，在命令行中输入 pip install PyMySQL 并按回车键来安装 PyMySQL 模块，等待提示 Successfully installed PyMySQL 后即可使用。关于 pip 的使用方式，在前文已经详细讲过，这里不再赘述。

1. 连接 MySQL 数据库

使用 Python 连接数据库前，先要确保数据库已经被正确创建，可以使用 Navicat for MySQL 创建新的数据库，这里使用前文已经创建过的数据库 newdatabase。连接数据库的代码如下：

```
#第 3 章//mysqls.py
import pymysql                              #导入 pymysql 模块
db = pymysql. connect ( user = " root", password = " 123456", host = " localhost", database =
"newdatabase")                              #连接数据库
cur = db.cursor()                           #创建一个游标对象
cur.execute("SELECT VERSION()")             #执行 SQL 语句
data = cur.fetchone()                       #获取单条数据
print("Database version: % s" % data)
db.close()

#输出结果为 Database version:8.0.23
```

PyMySQL 模块中的 connect()方法用于连接 MySQL 数据库，其包含的参数如下。

(1) host：要连接的主机地址。

(2) user：用于登录的数据库用户。

(3) password：数据库密码。

(4) database：要连接的数据库。

(5) port：数据库的端口，MySQL 默认端口为 3306。

(6) unix_socket：选择是否要用 unix_socket 而不是 TCP/IP。

(7) charset：字符编码。

(8) sql_mode：数据库模型。

(9) read_default_file：从默认配置文件(my.ini 或 my.cnf)中读取参数。

(10) conv：转换字典。

(11) use_unicode：是否使用 unicode 编码。

(12) cursorclass：选择 Cursor 类型。

(13) init_command：连接建立时运行的初始语句。

(14) connect_timeout：连接超时时间。

(15) ssl：使用 ssl 连接。

(16) read_default_group：要从配置文件中读取的组。

(17) autocommit：是否自动提交事务。

(18) db：同 database，为了兼容 MySQLdb。

(19) passwd：同 password，为了兼容 MySQLdb。

(20) local_infile：是否允许载入本地文件。

(21) max_allowed_packet：限制 LOCAL DATA INFILE 的大小。

(22) bind_address：当客户有多个网络接口时，指定一个连接到主机。

connect()方法的参数比较多，所以在定义参数时最好使用关键字参数。连接数据库后，创建一个游标对象。游标实际上是一种能从包括多条数据记录的结果集中每次提取一条记录的机制。游标可以被看作一个查询结果集(可以是零条、一条或由相关的选择语句检索出的多条记录)和结果集中指向特定记录的游标位置组成的一个临时文件，提供了在查询结果集中向前或向后浏览数据、处理结果集中数据的能力。有了游标，用户就可以访问结果集中任意一行数据，在将游标放置到某行后，可以在该行或从该位置的行块上执行操作。

使用 execute()方法执行了 SQL 语句 SELECT VERSION()，该方法包含两个参数，分别为 SQL 语句及 args 参数，execute()方法中 sql 的占位符是用小括号()括起来的占位符，示例代码如下：

```
...
cur.execute("insert into newtable(username) values ( % s)",val)
...
```

args 一般是 list 或 tuple 格式，如果只有一个参数，则可以直接传入。fetchone()方法用于返回数据集中的一条元素，与其同类型的还有 fetchall()，用于返回全部记录。fetchmany(*n*)返回 *n* 条记录。

2. 创建表

创建表前需要确保已经连接至数据库，使用 execute()方法执行创建表的 SQL 语句，下面创建一个表并命名为 news，用于存储新闻文章，字段分别为 id、title、bodys，数据类型分别为 int、varchar(45)、varchar(100)，其中 id 为主键并且自动增长，title 与 bodys 设置为 utf8 编码，示例代码如下：

```
# 第 3 章//mysqls.py
import pymysql
db = pymysql.connect(user = "root", password = "123456", host = "localhost", database =
"newdatabase")                                  # 连接数据库
cur = db.cursor()
cur.execute("DROP TABLE IF EXISTS news")    # 如果表存在则删除
sql = """
CREATE TABLE news (
    id int NOT NULL AUTO_INCREMENT,
    title varchar(45) CHARACTER SET utf8 COLLATE utf8_unicode_ci NULL DEFAULT NULL,
    bodys varchar(100) CHARACTER SET utf8 COLLATE utf8_unicode_ci NULL DEFAULT NULL,
    PRIMARY KEY (id) USING BTREE
) ENGINE = InnoDB CHARACTER SET = utf8 COLLATE = utf8_unicode_ci
```

```
"""
cur.execute(sql)                #执行 SQL 语句
db.close()
```

3. 插入数据

往上文创建的表中插入数据,示例代码如下:

```
#第 3 章//mysqls.py
import pymysql
db = pymysql.connect(user = "root", password = "123456", host = "localhost", database =
"newdatabase")            #连接数据库
cur = db.cursor()
args = ("标题","内容")
sql = "INSERT INTO news (title,bodys) VALUES (%s,%s)"
cur.execute(sql,args)     #执行 SQL 语句
db.commit()               #手动提交到数据库执行
db.close()
```

如果连接数据库时没有指定自动提交事务,则需要在执行代码后手动提交,即使用commit()方法。

4. 查询数据

使用 fetchone()、fetchall()、fetchmany(*n*)处理查询出的结果集,fetchone()方法用于返回数据集中的一条元素,fetchall()方法用于返回全部记录,fetchmany(*n*)方法用于返回 *n* 条记录。使用 fetchall()处理数据集的示例代码如下:

```
#第 3 章//mysqls.py
import pymysql
db = pymysql.connect(user = "root", password = "123456", host = "localhost", database =
"newdatabase")              #连接数据库
cur = db.cursor()
sql = "SELECT * FROM news"
cur.execute(sql)            #执行 SQL 语句
db.commit()                 #手动提交到数据库执行
result = cur.fetchall()     #处理数据集
for i in result:
    print(i)
db.close()

#输出结果为
(4, '5435', '534543')
(6, '标题', '内容')
(7, '标题 1', '内容 1')
(8, '标题', '内容')
```

5. 更新数据

更新 id 为 4 的数据,并将 title 修改为 aaa,将 bodys 修改为 bbb,示例代码如下:

```
#第 3 章//mysqls.py
import pymysql
db = pymysql. connect(user = "root", password = "123456", host = "localhost", database =
"newdatabase")          #连接数据库
cur = db.cursor()
args = ("aaa","bbb",4)
sql = "UPDATE news set title = %s,bodys = %s where id = %s"
cur.execute(sql,args)  #执行 SQL 语句
db.commit()             #手动提交到数据库执行
print("更新成功")
db.close()

#输出结果为更新成功
```

6. 删除数据

删除 id 为 4 的数据,示例代码如下:

```
#第 3 章//mysqls.py
import pymysql
db = pymysql. connect(user = "root", password = "123456", host = "localhost", database =
"newdatabase")          #连接数据库
cur = db.cursor()
args = 4
sql = "DELETE FROM news where id = %s"
cur.execute(sql,args)  #执行 SQL 语句
db.commit()             #手动提交到数据库并执行
print("删除成功")
db.close()

#输出结果为删除成功
```

7. 执行事务

PyMySQL 中可使用 commit()及 rollback()两种方法来完成一个事务的处理,示例代码如下:

```
#第 3 章//mysqls.py
import pymysql
db = pymysql. connect(user = "root", password = "123456", host = "localhost", database =
"newdatabase")          #连接数据库
cur = db.cursor()
sql = "DELETE FROM news where id = 4"
sql2 = "DELETE FROM news where id = 6"
try:
    cur.execute(sql)
    cur.execute(sql2)
    db.commit()
except:
    db.rollback()
db.close()
```

在上面的代码中多个 SQL 语句同时执行，要成功则一起成功，如果有一个执行不成功，则全部不成功，执行不成功需使用 rollback()方法进行回滚。回滚是指将数据恢复到上一次状态的行为。

3.4 MongoDB 简介及安装

3.4.1 MongoDB 简介

MongoDB 属于非关系型数据库，它由 C++语言编写，旨在为 Web 应用提供可扩展的高性能数据库存储及解决方案。MongoDB 是开源的产品，与 MySQL 类似，除企业版本及一些增值服务以外，MongoDB 本身对用于商用是不收取费用的。

MongoDB(来自于英文单词 Humongous，中文含义为“庞大”)是可以应用于各种规模的企业、各个行业及各类应用程序的开源数据库。作为一个适用于敏捷开发的数据库，MongoDB 的数据模式可以随着应用程序的发展而灵活地更新。与此同时，它也为开发人员提供了传统数据库的功能：二级索引、完整的查询系统及严格一致性等。MongoDB 能够使企业更加具有敏捷性和可扩展性，各种规模的企业都可以通过使用 MongoDB 来创建新的应用，提高与客户之间的工作效率，加快产品上市时间，以及降低企业成本。

MongoDB 是专为可扩展性、高性能和高可用性而设计的数据库。它可以从单服务器部署并扩展到大型、复杂的多数据中心架构。利用内存计算的优势，MongoDB 能够提供高性能的数据读写操作。MongoDB 的本地复制和自动故障转移功能使应用程序具有企业级的可靠性和操作灵活性。

3.4.2 MongoDB 特性

1. 文档型数据库

文档型数据库存储格式是版本化的文档、半结构化的文档及特定格式存储，例如 JSON。文档型数据库允许嵌套键值，但比键值数据库的查询效率更高。文档型数据库操作起来也比较简单和容易。

2. 高可用

MongoDB 提供了数据副本，当发生硬件故障或者服务中断时，可以从副本恢复数据，并能自动进行故障转移。运维简单，故障自动切换。

3. 高性能

mmapv1、wiredtiger、mongorocks(rocksdb)、in-memory 等多种存储引擎支持满足各种场景需求。

4. 易部署

MongoDB 支持多种平台，如 Windows、Mac OS、Linux 等。官方提供非常详细的部署资料，多数云平台直接支持 MongoDB。可以非常方便地创建单例 MongoDB 及 MongoDB 集群。

5. 模式自由

模式自由(Schema-Free)，意味着对于存储在 MongoDB 数据库中的文件，我们不需要

知道它的任何结构定义。如果需要，则完全可以把不同结构的文件存储在同一个数据库中。

6. 面向集合

所谓“面向集合”(Collection-Oriented)，意思是数据被分组存储在数据集中，被称为一个集合(Collection)。每个集合在数据库中都有一个唯一的标识名，并且可以包含无限数目的文档。集合的概念类似于关系数据库(RDBMS)里的表(table)，不同的是它不需要定义任何模式，能够快速识别数据库内大数据集中的热数据，提供一致的性能改进。

7. 多语言支持

支持 Perl、PHP、Java、C#、JavaScript、Ruby、C 和 C++语言的驱动程序，MongoDB 提供了当前所有主流开发语言的数据库驱动包，开发人员使用任何一种主流开发语言都可以轻松编程，实现访问 MongoDB 数据库。

3.4.3 MongoDB 安装

1. Windows 平台下 MongoDB 的安装

MongoDB 的官方网站为 https://www.mongodb.com，与 MySQL 相同的是，MongoDB 同样也分为社区版与企业版，社区版是免费开源的，企业版包含了一些增值服务。日常开发选择社区版就足够用了。

打开 MongoDB 官网→Software 单击 Community Server 菜单即可显示社区版 MongoDB 的下载页面，如图 3-40 所示。在页面的 MongoDB Community Server 这一栏右侧可以选择 MongoDB 的版本进行下载，在本书的写作期间，MongoDB 的最高版本为 4.4.4，平台选择 Windows，选择 msi 安装包，单击 Download 按钮即可开始 MongoDB 的下载。

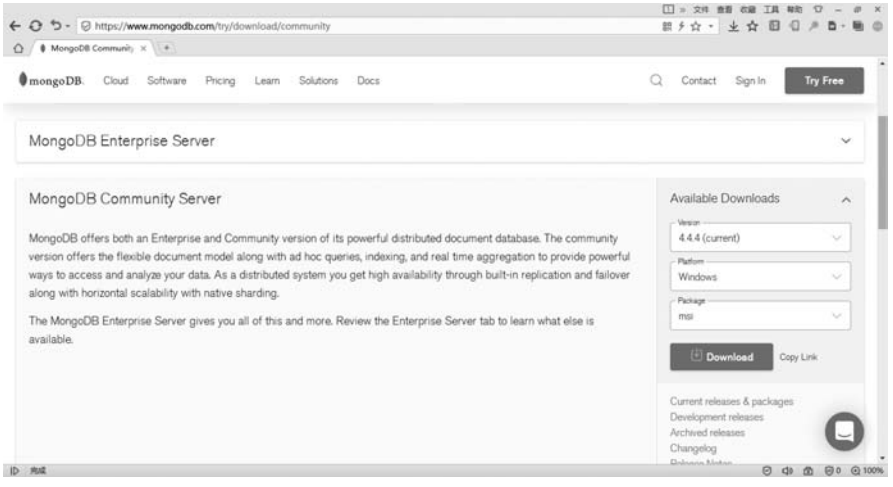

图 3-40 下载 MongoDB

双击下载好的 MongoDB 安装包即可开始对 MongoDB 进行安装，前两页单击 Next 按钮进行下一步。在 Choose Setup Type 页面 MongoDB 提供了安装模式的选择，该页面有两个选项，一个是完全安装，另一个是自定义安装，如图 3-41 所示。

这里选择 Custom，即自定义安装，单击 Custom 按钮即可显示自定义安装界面，如图 3-42 所示。

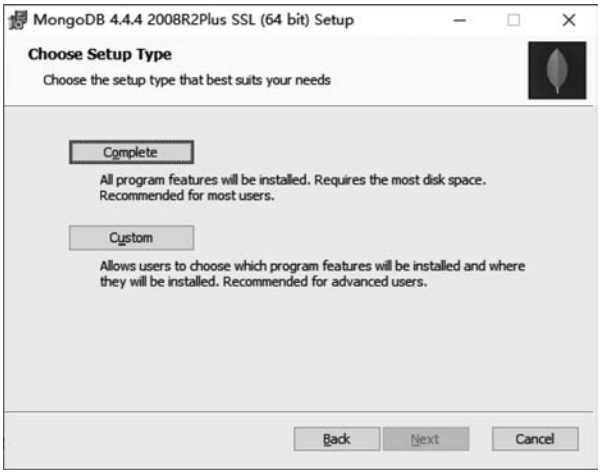

图 3-41 选择安装模式

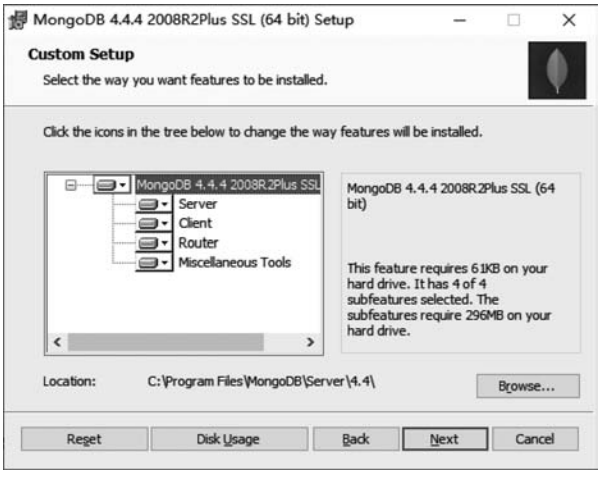

图 3-42 自定义安装

在自定义安装界面可见准备安装到计算机上的程序包，共含 4 部分，Server 为 MongoDB 的服务器端主程序，Client 为 MongoDB 的客户端工具，Router 向外提供应用访问的接口，Miscellaneous Tools 为 MongoDB 的一些其他工具。这里默认为都选中，按默认值安装即可。

在 Location 处选择要安装的路径，此处选择任意目录即可，切记目录中不要有中文字符及其他特殊字符，例如圆角空格等，因为这些字符有可能会产生一些意料之外的错误。选择好目录以后单击 Next 按钮，来到服务配置界面，如图 3-43 所示。此时默认勾选了 Install MongoD as Service，即将 MongoDB 作为服务安装到 Windows 操作系统上，这样做的好处是随着系统的启动，MongoDB 也跟着启动，所以此处要勾选。

在其下方有两个选项，分别为 Run service as Network Service user(使用网络用户运行服务)，以及 Run service as a local or domain user(以本地域或用户身份运行服务)，此处使用默认选择，即 Run service as Network Service user。

(1) Service Name 为设置 MongoDB 服务的服务名称,此处默认为 MongoDB。

(2) Data Directory 为设置数据库文件的存储目录。

(3) Log Directory 为设置日志文件的目录。

设置完毕后单击 Next 按钮进入 MongoDB Compass 安装界面,如图 3-44 所示。MongoDB Compass 为 MongoDB 的图形化工具,此处勾选 Install MongoDB Compass,单击 Next 按钮进入下一步。继续单击 Install 按钮开始 MongoDB 的安装,安装时间比较长,需要耐心等待进度条读取完毕,然后单击 Finish 按钮即完成了 MongoDB 的安装。

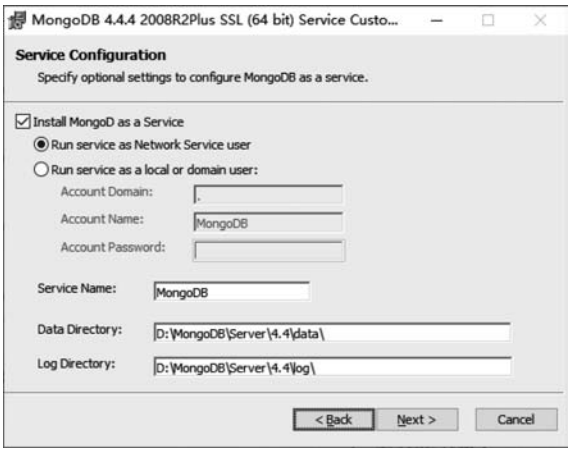

图 3-43 配置 MongoDB

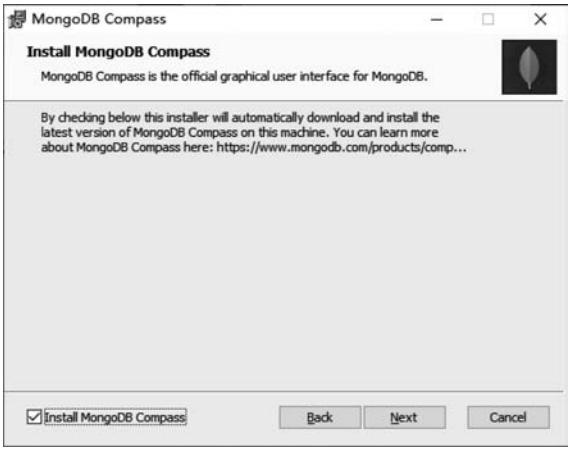

图 3-44 安装 MongoDB Compass

2. Linux 平台下 MongoDB 的安装

打开 MongoDB 的官方网站,网址为 https://www.mongodb.com,单击 Software→Community Server 链接进入 MongoDB Community Server 下载页,因为笔者的 Linux 操作系统是 CentOS 7.9,所以在右侧 Available Downloads 面板中的 Platform 栏选择 RedHat/CentOS 7.0,注意不要选择 RedHat/CentOS 7.2 s390x。在 Package 栏选择 server,如图 3-45 所示。

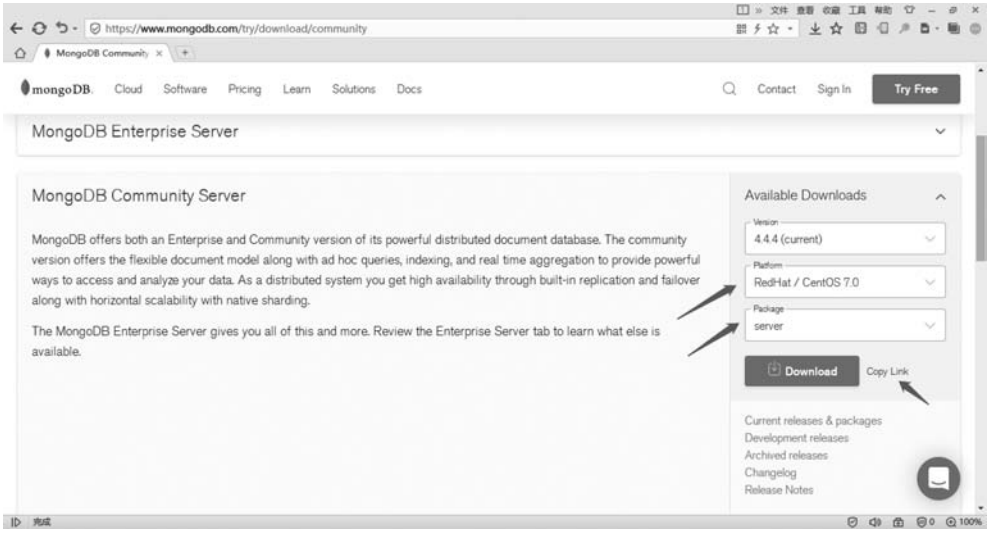

图 3-45　MongoDB 下载页面

如果不知道自己安装的 CentOS 的具体版本是多少，可以使用如下命令在命令行执行，命令如下：

```
cat /etc/redhat-release

#输出结果为 CentOS Linux release 7.9.2009 (Core)
```

选择好平台和包后，单击 Download 旁边的 Copy Link 链接，复制 rpm 包链接地址。地址为 https://repo.mongodb.org/yum/redhat/7/mongodb-org/4.4/x86_64/RPMS/mongodb-org-server-4.4.4-1.el7.x86_64.rpm。获取 rpm 包链接后，在 CentOS 命令行中执行如下命令

```
wget -i -c https://repo.mongodb.org/yum/redhat/7/mongodb-org/4.4/x86_64/RPMS/mongodb-org-server-4.4.4-1.el7.x86_64.rpm
```

该命令会下载一个 mongodb-org-server-4.4.4-1.el7.x86_64.rpm 的包文件，当文件下载完毕后，安装该包，命令如下：

```
yum -y install mongodb-org-server-4.4.4-1.el7.x86_64.rpm
```

等待文件安装完毕，出现 Complete 后表示 MongoDB Server 安装完成了。启动 MongoDB 服务命令如下：

```
systemctl start mongod.service
```

此时如果没有报错，则表示 MongoDB 已经启动了，查看 MongoDB 运行状态的命令如下：

```
systemctl status mongod.service
```

可以看到此时 MongoDB 已经正常运行了，如图 3-46 所示。

```
[root@localhost ~]# systemctl status mongod.service
● mongod.service - MongoDB Database Server
   Loaded: loaded (/usr/lib/systemd/system/mongod.service; enabled; vendor preset: disabled)
   Active: active (running) since Sat 2021-04-03 06:24:04 EDT; 1min 37s ago
     Docs: https://docs.mongodb.org/manual
  Process: 6953 ExecStart=/usr/bin/mongod $OPTIONS (code=exited, status=0/SUCCESS)
  Process: 6951 ExecStartPre=/usr/bin/chmod 0755 /var/run/mongodb (code=exited, status=0/SUCCESS)
  Process: 6949 ExecStartPre=/usr/bin/chown mongod:mongod /var/run/mongodb (code=exited, status=0/SUCCESS)
  Process: 6948 ExecStartPre=/usr/bin/mkdir -p /var/run/mongodb (code=exited, status=0/SUCCESS)
 Main PID: 6956 (mongod)
   CGroup: /system.slice/mongod.service
           └─6956 /usr/bin/mongod -f /etc/mongod.conf
```

图 3-46　MongoDB 的运行状态

接下来安装 MongoDB Shell，再回到 MongoDB 下载页面，此时修改 Available Downloads 面板中的 Package 选项选择 shell(rpm)，如图 3-47 所示。

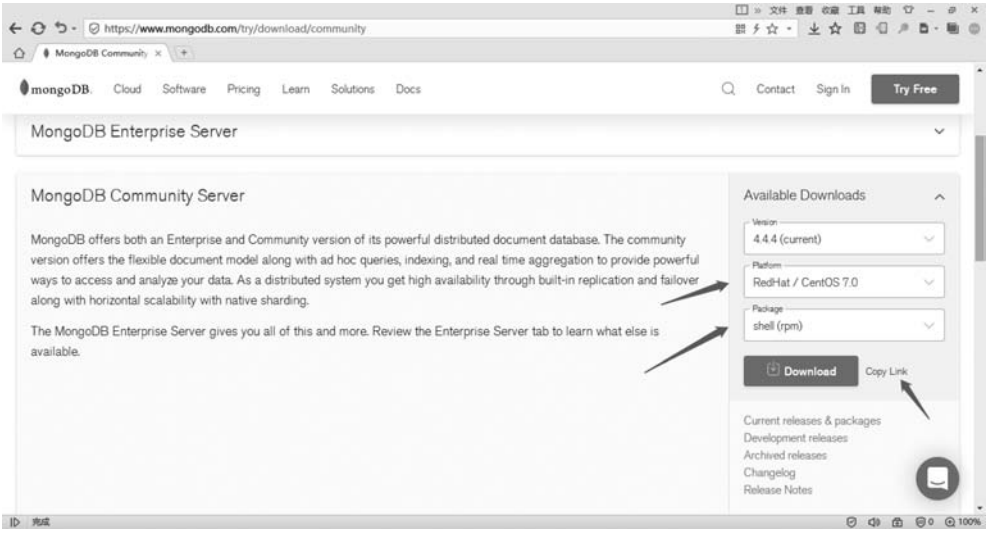

图 3-47　选择 shell(rpm)

然后单击 Copy Link 连接复制 shell(rpm)包下载网址，其下载网址为 https://repo.mongodb.org/yum/redhat/7/mongodb-org/4.4/x86_64/RPMS/mongodb-org-shell-4.4.4-1.el7.x86_64.rpm，回到 CentOS 下载包，执行命令如下：

```
wget -i -c https://repo.mongodb.org/yum/redhat/7/mongodb-org/4.4/x86_64/RPMS/mongodb-org-shell-4.4.4-1.el7.x86_64.rpm
```

等待下载完毕后会获得一个名为 mongodb-org-shell-4.4.4-1.el7.x86_64.rpm 的文件，安装该文件的命令如下：

```
yum -y install mongodb-org-shell-4.4.4-1.el7.x86_64.rpm
```

等待文件安装完毕后，即可运行 MongoDB Shell 了，进入 Shell 的命令如下：

```
mongo
```

显示所有数据库命令如下：

```
show dbs

#输出结果为
admin  0.000GB
config 0.000GB
local  0.000GB
```

至此完成了 MongoDB 在 Linux 系统上的安装。因本书主要讲解 Python 编程语言，所以不再详细展开，如需对 MongoDB 进行进一步的设置，则可参考 MongoDB 官方文档。

3.4.4　MongoDB 可视化工具

因为在安装 MongoDB 时勾选了 Install MongoDB Compass 选项，所以在安装程序运行完后会弹出 MongoDB Compass 界面，MongoDB Compass 是 MongoDB 官方提供的免费的可视化工具。初次打开 MongoDB Compass 会弹出 Privacy Settings 选项卡，即隐私设置选项卡，如图 3-48 所示。

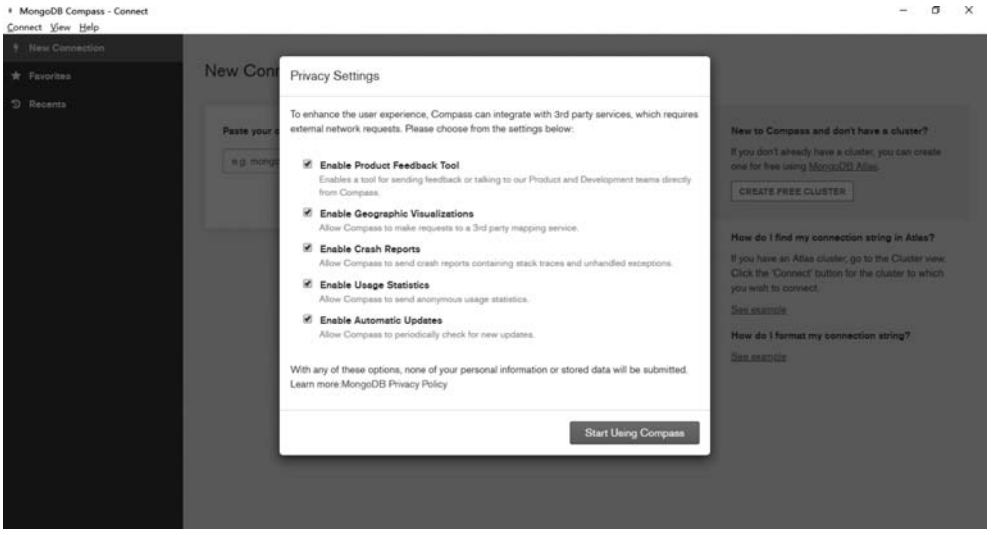

图 3-48　MongoDB Compass 隐私设置

该界面显示的大意为“为了增强用户体验，Compass 需要与外部网络的第三方服务集成请求。请从下面的设置中选择。”

(1) Enable Product Feedback Tool：允许向开发团队发送反馈信息。

(2) Enable Geographic Visualizations：启用地理可视化。

(3) Enable Crash Reports：允许发送错误报告。

(4) Enable Usage Statistics：允许发送匿名的使用统计信息。

(5) Enable Automatic Updates：启动自动更新。

可以根据自身情况选择需要开启或者关闭的信息，在这里笔者选择允许所有选项，然后单击 Start Using Compass 按钮即可开始使用 MongoDB Compass。

单击左侧菜单中的 New Connection 按钮，用来连接 MongoDB 服务器，此时 MongoDB Compass 界面上默认采取的是字符串的连接方式，对于刚接触 MongoDB 的读者来讲这样的连接方式看上去会有点陌生，可以单击 Fill in connection fields individually 链接切换到通过填写参数的方式来连接 MongoDB 服务器，如图 3-49 所示。

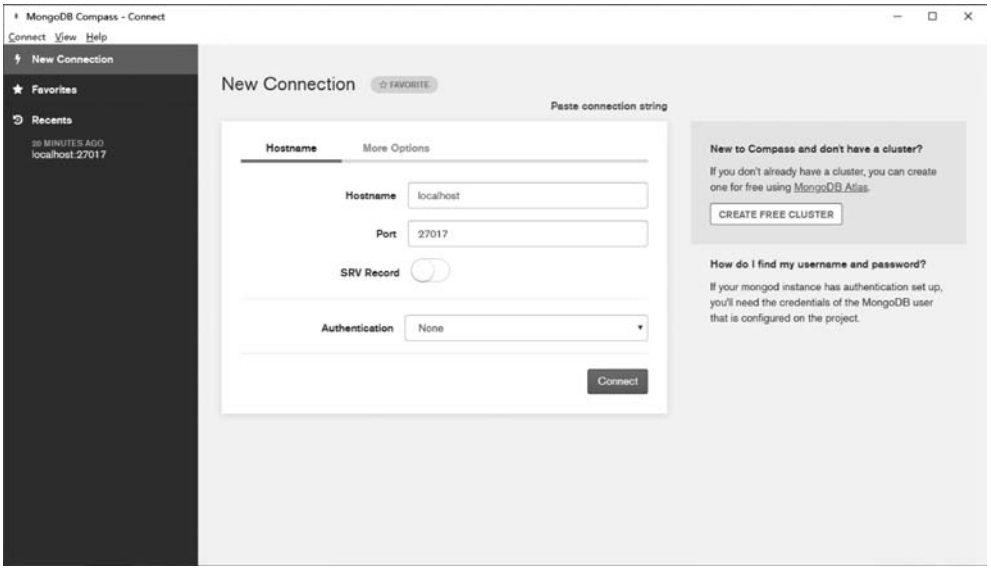

图 3-49　切换至填写参数的模式

该界面包含两个 Tab 选项，分别为 Hostname 与 More Options，Hostname 包含了 Hostname、Port、SRV Record 及 Authentication 参数。其中 Hostname 表示要连接的 MongoDB 服务器地址，默认为 localhost，即为本机。Port 表示要连接的 MongoDB 服务器的端口，MongoDB 的默认端口号为 27017。SRV Record 表示使用 SRV 的方式进行连接，选择此处后，Hostname 需要填写 SRV 字符串，而不能使用 IP 地址了，否则会提示 URI does not have hostname, domain name and tld 错误。Authentication 表示使用的加密方式，当前 MongoDB 的加密方式有 5 种，分别为 Username/Password、SCRAM-SHA-256、Kerberos、LDAP、X.509，其中 Username/Password、SCRAM-SHA-256 是项目中使用得比较多的验证方式。之前的安装没有设置 MongoDB 的权限，所以此处 Authentication 选择 None。

More Options 中包含了 Replica Set Name、Read Preference、SSL、SSH Tunnel 参数，其中 Replica Set Name 表示副本集的名称。Read Preference 为读取的首选项，可用于设置数据库的读写分离，以及故障转移等操作。该参数下有 Primary（主节点）、Primary Preferred（首选主节点）、Secondary（从节点）、Secondary Preferred（首先从节点）、Nearest（邻近节点）几个选项。SSL 参数为 MongoDB 的网络通信进行加密，其包含 4 种方式，System CA/Atlas Deployment、Server Validation、Server and Client Validation、Unvalidated。SSH Tunnel 参数为通过远程服务器连接 MongoDB 服务器，该参数包含 2 个

选项，分别为 Use Password 与 Use Identity File。

因为连接的是本机的 MongoDB 单例服务器，所以 More Options 无须进行配置，Hostname 使用默认设置即可，单击 Connect 按钮连接到本机的 MongoDB 服务器，如图 3-50 所示。

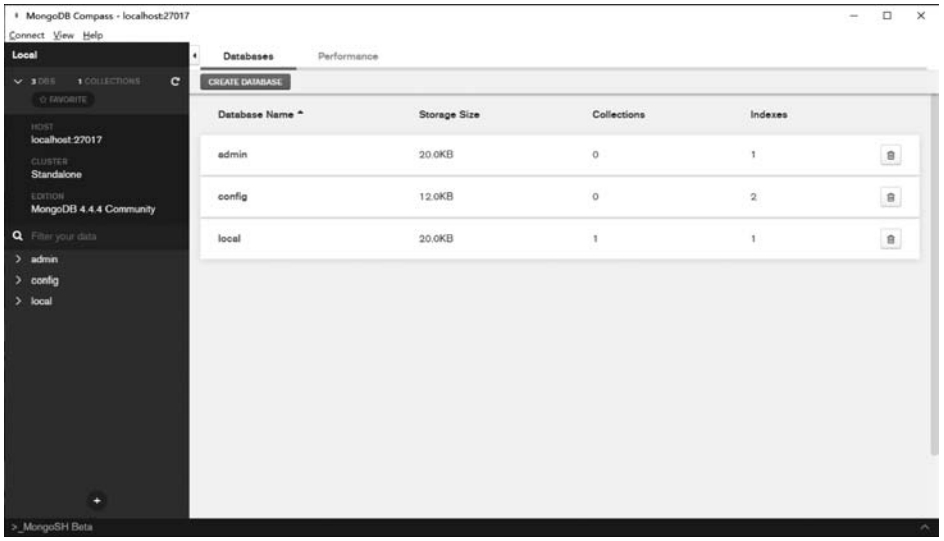

图 3-50 连接已安装的 MongoDB 服务器

单击 CREATE DATABASE 按钮后可创建新的数据库，在 Database Name 参数下输入 newdatabase，在 Collection Name 参数下输入 newcollection。Capped Collection 参数用于设置为固定集合。固定集合的意思是性能出色，有固定大小的集合。当集合空间使用完后，再插入的数据将会从第 1 条数据开始覆盖，此处无须勾选固定集合。Use Custom Collation 参数为使用用户自定义的方式进行排序，此处无须勾选，如图 3-51 所示。单击 CREATE DATABASE 按钮就完成了 MongoDB 的数据库的创建。

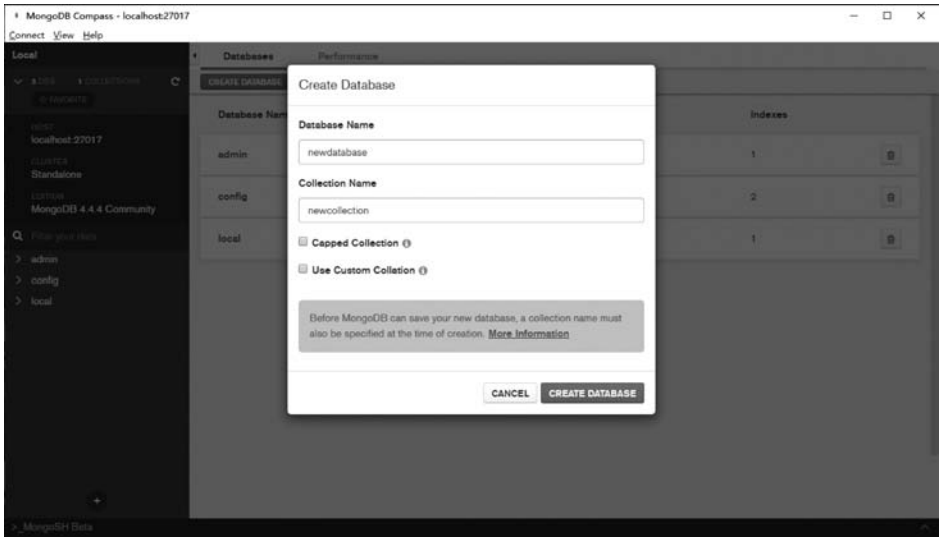

图 3-51 创建 MongoDB 数据库

在 MongoDB Compass 默认界面可以看到刚才创建的数据库 newdatabase，单击 newdatabase 链接，进入 Collections 界面可以看到刚才创建的 newcollection。继续单击 newcollection 链接即可进入 newcollection 集合内，在此可以添加任何类型的数据。单击 ADD DATA 添加一条新数据，在 ADD DATA 下有两个选项，分别为 Import File 与 Insert Document。Import File 从文件导入数据，支持 JSON 与 CSV 两种格式。Insert Document 则直接在 MongoDB Compass 中添加数据，此处选择 Insert Document，单击 Insert Document 按钮进入添加数据页面，如图 3-52 所示。在 VIEW 处选择列表视图，将鼠标移动到_id 字段之上，在前方会出现一个加号＋按钮，单击加号按钮便可以添加一个新的字段。

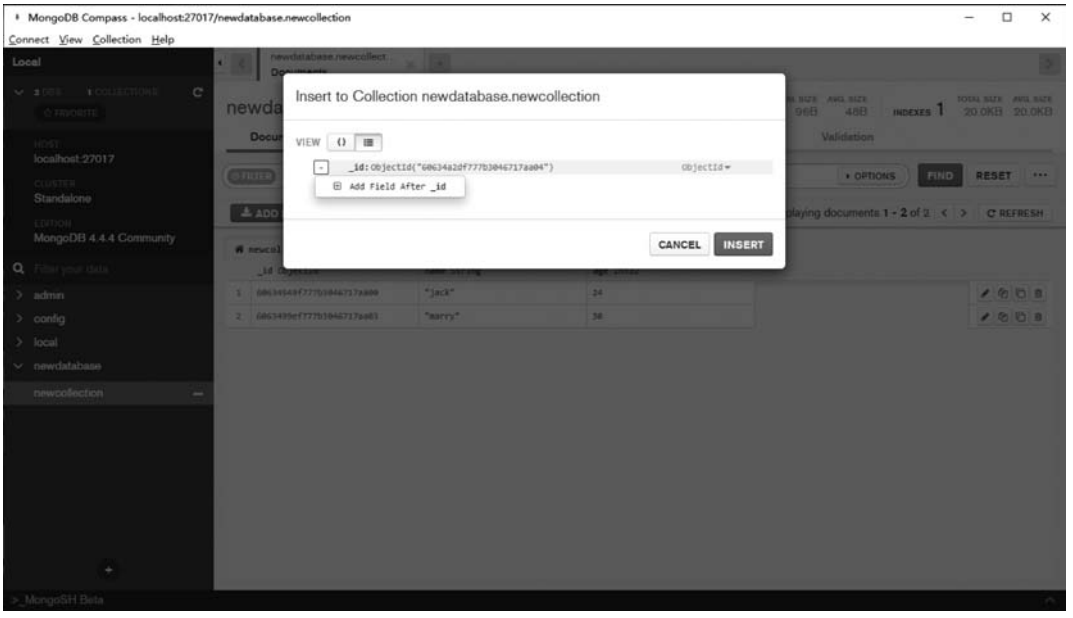

图 3-52　添加 Document

使用此方法在 newcollection 中插入两条数据，如表 3-3 所示。

表 3-3　添加的内容

字段名	age	name
数据 1	24	jack
数据 2	30	marry
类型	Int32	String

将上面内容添加到集合中后，在 MongoDB Compass 界面的 newcollection 集合下就可以看到添加的两条内容了，如图 3-53 所示。

在当前页有 3 种视图选项按钮，分别是列表式、结构式及表格式，可以根据自身习惯选择不同的表现样式。对于列表式与结构式两种表现样式，当鼠标移动到内容上时，即会弹出功能菜单列表，包括编辑、复制、克隆及删除。对于表格式，功能菜单直接显示在每行内容的最后面，使用功能菜单可以非常方便地对数据进行编辑、复制、克隆及删除。

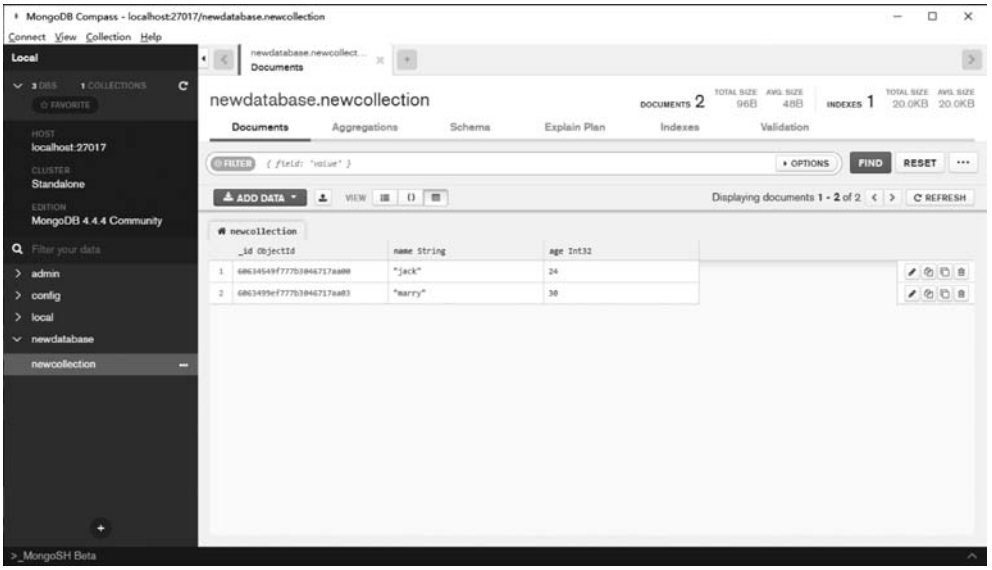

图 3-53　已添加的内容

3.4.5　MongoDB 基础

6min

1. MongoDB 数据库与集合

MongoDB 与 MySQL 虽然属于不同类型的数据库，MongoDB 为非关系型数据库，MySQL 为关系型数据库，但是在概念上却极为相似，在 MongoDB 中同样有数据库的概念，其他诸如集合、文档、字段、索引等对于 MySQL 都有一一对应的概念，具体如表 3-4 所示。MongoDB 内可存储多个数据库，每个数据库可包含多个集合，而每个集合又可以包含多个文档。

表 3-4　MongoDB 与 MySQL 对应的概念

mysql	mongodb	说　明
database	database	数据库
table	collection	数据表/集合
row	document	记录行/文档
column	field	字段
index	index	索引
primary key	primary key	主键，MongoDB 自动将_id 字段设置为主键

2. MongoDB 格式及数据类型

MongoDB 文档支持 JSON 与 BSON 两种数据格式，JSON（JavaScript Object Notation）是一种轻量的数据交换格式，因为轻量，因此它易于理解、易于解析、易于记忆，但 JSON 的数据类型有限，只有 null、布尔型、数字、字符串、数组及对象这几种类型，所以 JSON 有一定的局限性。JSON 通常的格式代码如下：

```
{
    "user": "jack",
    "pass": "123456",
    "address": [{
        "city": "wuhan",
        "code": "434000",
        "tel": "123456"
    }]
}
```

BSON(Binary Serialized Document Format)也是一种数据交换格式,主要用于MongoDB数据库中的数据存储和网络传输,它是一种二进制表示形式。相较于JSON,BSON支持的数据类型更加丰富,且BSON主要用于提高存储和扫描效率。BSON常见格式代码如下:

```
{
    user:"jack",
    pass:"123456",
    nowday: new Date('Jun 23, 2021'),
    otherday: new Date('Jun 07, 2022'),
    address:{
        city:"wuhan",
        tel:"123456",
        code:434000
    } ,
    scores:[
        {"name":"english","grade:3.0},
        {"name":"chinese","grade:2.0}
    ]
}
```

MongoDB将数据记录存储为BSON文档,BSON文档由一个有序的元素列表构成。每个元素由一个字段名、一种类型和一个值组成,字段名为字符串。BSON类型如表3-5所示。

表3-5 MongoDB的数据类型

类　型	说　明
Array	数组类型
Binary	二进制类型
Boolean	布尔类型
Date	定长字符串,最大长度为255字节
Decimal128	128位IEEE 754-2008浮点数; Binary Integer Decimal的变体
Double	浮点数
Int32	32位整数
Int64	64位整数
MaxKey	最大值

续表

类　型	说　明
MinKey	最小值
Null	Null
Object	对象
ObjectId	ID 对象
BSONRegExp	BSON 正则表达式
String	字符串类型
BSONSymbol	符号
BSONMap	Map
Timestamp	时间戳
Undefined	Undefined

BSON 内没有自增字段，自增需要通过代码实现，且 BSON 文档的最大大小为 16MB 字节，最大文档的大小有助于确保单个文档不能使用过多的内存，或者在传输期间不能使用过多的带宽。为了存储大于最大大小的文档，MongoDB 提供了 GridFS API。

在 MongoDB 文档内有一个特殊的字段为_id，该字段默认由 MongoDB 自动生成，该字段具有唯一性，且默认被设置为主键。该字段总是文档中的第 1 个字段，如果服务器首先接收到一个没有_id 字段的文档，则服务器将把该字段移到开头。_id 字段可以包含除数组之外的任何数据类型的值。

MongoDB 创建的默认_id 通常为 6065400d182c9cf6587975dc 样式，该字段中包含了 24 位十六进制数，也就是 12 字节（每字节由 2 个十六进制数组成），6065400d182c9cf6587975dc 对应的划分如表 3-6 所示。

表 3-6　ObjectID 的划分

60	65	40	0d	18	2c	9c	f6	58	79	75	dc
时间戳				机器码			PID		计数器		

时间戳也就是从 1970 年 1 月 1 日（UTC/GMT 的午夜）开始所经过的秒数，机器码则表示运行 MongoDB 的计算机，PID 表示生成次_id 的进程，计数器是由一个随机开始的计数器生成的值。MongoDB 通过如此复杂的生成方式，既可以保证_id 字段的唯一性，又可以通过该字段所包含的含义进行查询及排序，非常方便。也可以根据自己的需要定制符合需求的生成规则。

3.5　MongoDB 操作语法

21min

打开 MongoDB Compass，在界面左下角有一个 MongoSH Beta 链接，单击该链接即可进入 MongoDB Shell，如图 3-54 所示。

MongoDB 除了可以使用 MongoDB Compass 可视化操作以外，还有很多命令的操作方式，该方式可以在 MongoDB Shell 下执行，且其他语言使用 MongoDB 时也可以通过调用命令的方式对 MongoDB 进行操作。下面的命令都可以在 MongoSH Beta 中执行。

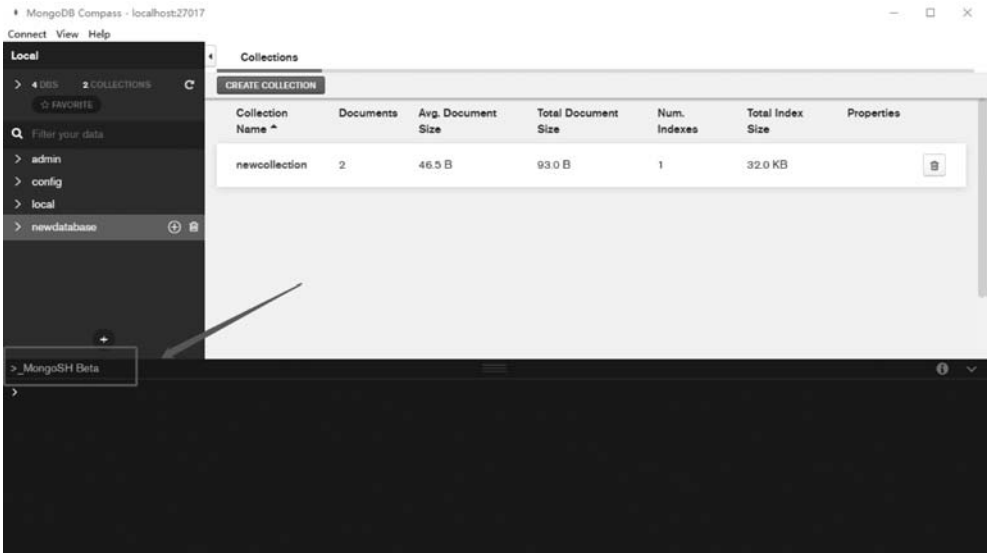

图 3-54　打开 MongoDB Compass 中的 MongoSH Beta

1. 创建数据库

创建一个新的 MongoDB 数据库 newdt 的代码如下：

```
use newdt

#输出结果为'switched to db newdt'
```

上面的代码可以用于创建一个名为 newdt 的新数据库。注意，如果 newdt 已经存在，则直接使用原数据库而不再创建新数据库。

可以使用 db 命令查看当前数据库的名称，代码如下：

```
db

#输出结果为 newdt
```

如果想查看所有的数据库，则可以使用 show dbs 命令来查看，代码如下：

```
show dbs

#输出结果为
admin         65.5 kB
config        98.3 kB
local         73.7 kB
newdatabase   65.5 kB
```

刚才创建了新的数据库后，不论是使用 show dbs 命令还是在 Mongo Compass 内并没有显示刚创建的数据库 newdt，因为该数据库内没有数据，如果想显示它，需要向该数据库内插入一些数据。插入数据的示例代码如下：

```
db.newcoll.insert({"name":"jack"})

#输出结果为
{ acknowledged: true,
  insertedIds: { '0': ObjectId("606569d2aa45a9378c309883") } }
```

单击 MongoDB Compass 界面上的刷新按钮，即可看到新创建的数据库 newdt，如图 3-55 所示。

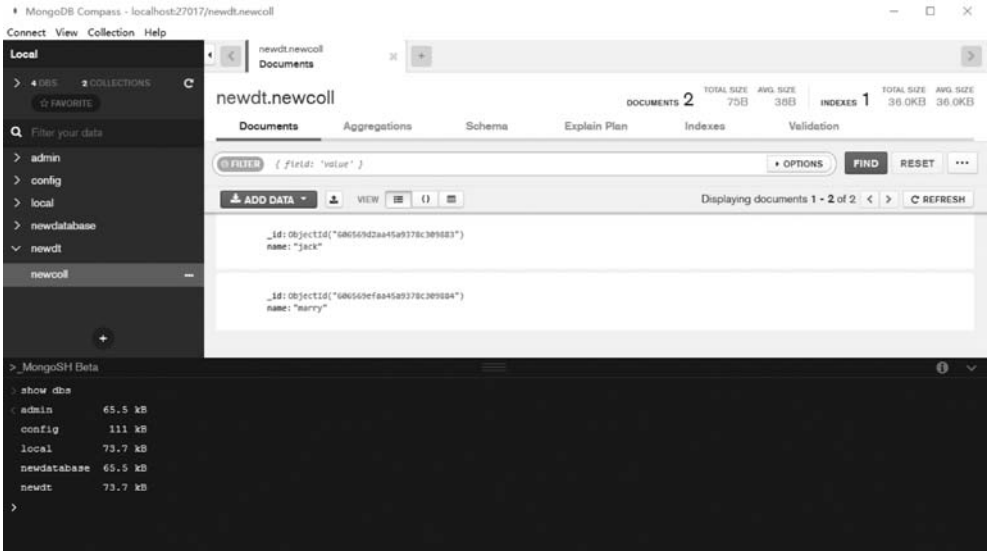

图 3-55　显示新创建的数据库

使用 show dbs 来查看，则会显示出创建的数据库 newdt，代码如下：

```
show dbs

#输出结果为
admin          65.5 kB
config         111 kB
local          73.7 kB
newdatabase    65.5 kB
newdt          73.7 kB
```

2. 删除数据库

删除数据库的语法格式如下：

```
db.dropDatabase()
```

删除某个数据库前，需要先使用该数据库，即 use 数据库名，然后使用删除命令。例如将前面创建的 newdt 数据库删除，先切换至该数据库，代码如下：

```
use newdt

#输出结果为'switched to db newdt'
```

执行删除命令，代码如下：

```
db.dropDatabase()

#输出结果为{ ok: 1, dropped: 'newdt' }
```

单击 MongoDB Compass 上的刷新按钮，则看不见数据库 newdt 了，如图 3-56 所示。使用 show dbs 命令来查看，也看不见 newdt 了，代码如下：

```
show dbs

#输出结果为
admin         65.5 kB
config        111 kB
local         73.7 kB
newdatabase   65.5 kB
```

数据库被删除后，其内的集合及文档都将一并被删除，删除数据库是比较危险的操作，在删除数据库前最好做好备份，以防不测。

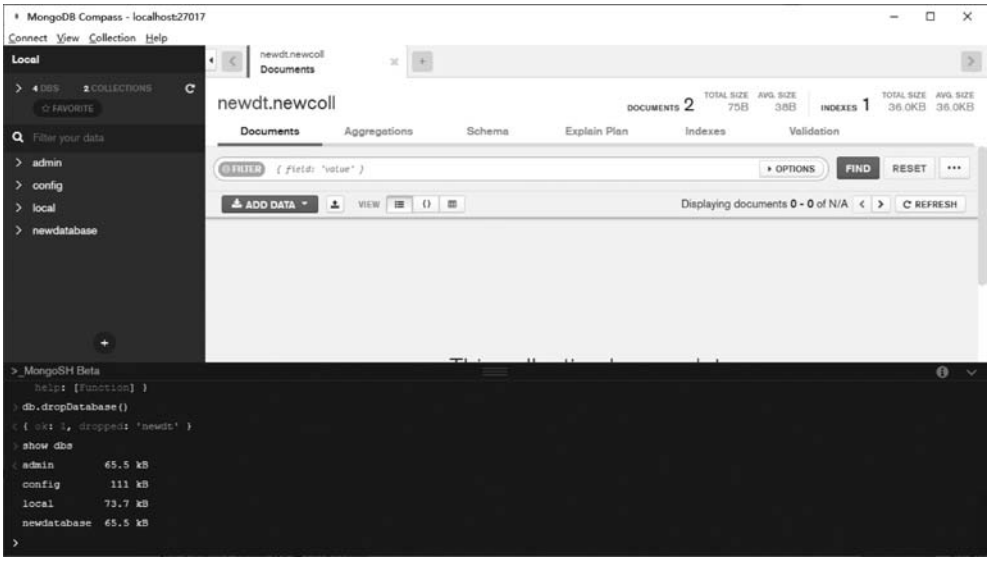

图 3-56　删除数据库

3. 创建集合

MongoDB 中使用 createCollection()方法来创建集合，其语法格式如下：

```
db.createCollection(name, options)
```

其中，name 为要创建的集合的名称，options 为可选参数。options 包含的参数如表 3-7 所示。

表 3-7 删除数据库

参 数	类 型	说 明
capped	boolean	如需创建固定集合，则该参数需设置为 true 且需要设置 size 参数的大小
autoIndexId	boolean	3.2 版本后不再支持该参数。当设置为 false 时，禁止对_id 字段自动创建索引
size	number	指定固定集合的最大字节数
max	number	指定固定集合中的最大文档数
storageEngine	document	仅支持 WiredTiger 存储引擎，允许用户配置存储引擎
validator	document	验证器，允许用户为集合指定验证规则
validationLevel	string	指定 MongoDB 在更新过程中对现有文档验证规则的严格程度
validationAction	string	对无效文档的提示形式
indexOptionDefaults	document	创建集合时为索引指定默认的配置
viewOn	string	源 collection 或源视图，指定依赖某个集合或视图而建的
pipeline	array	聚合管道，作为聚合管道片段的一部分，参与聚合操作
collation	document	指定集合的默认排序规则
writeConcern	document	写安全机制，用于控制写入安全的级别

使用 createCollection()方法在新创建的数据库 newdt 中创建一个新的集合 newcoll，代码如下：

```
use newdt
db.createCollection("newcoll")

#输出结果为{ ok: 1 }
```

此时已经完成了集合 newcoll 的创建，在 MongoDB Compass 内单击“刷新”按钮即可看到刚创建的数据库 newdt 及数据库内的集合 newcoll。

也可以使用 options 参数创建一个固定集合，示例代码如下：

```
use newdt
db.createCollection("newcoll2",{capped:true,size:6000000,max:10000})

#输出结果为{ ok: 1 }
```

此时创建了一个固定集合，集合的最大字节数为 6000000，最大文档数为 10000。在 MongoDB Compass 内单击“刷新”按钮即可看到刚才创建的固定集合 newcoll2。固定集合的意思是集合的大小被固定了，当集合空间使用完以后，再插入文档将会从第 1 条文档开始进行覆盖，如果指定了最大文档数，则同样当插入的文档数量超过了最大文档数，将会从第 1 条文档开始进行覆盖。固定集合就好比一个圆环，当空间使用完了以后，又回到起点。

4. 删除集合

使用 drop()方法来删除集合,其语法格式如下:

```
db.集合名称.drop()
```

例如要删除 newdt 数据下的 newcoll2 集合,代码如下:

```
use newdt
db.newcoll2.drop()

#输出结果为 true
```

5. 查看所有集合

使用 show collections 命令查看当前数据库下的所有集合,示例代码如下:

```
use newdt
db.createCollection("newcoll3")
show collections

#输出结果为
newcoll3
newcoll
```

6. 插入文档

使用 insert()方法来插入文档,insert()方法既可以插入单条文档,也可以插入多条文档。

其语法格式如下:

```
db.collection.insert(
   <document or array of documents>,
   {
     writeConcern: <document>,
     ordered: <boolean>
   }
)
```

document 为要插入的文档。writeConcern 为写入策略,默认值为 1,即要求确认写操作,当参数值为 0 时表示不要求确认写操作。ordered 指定是否按顺序写入,默认值为 true,即按顺序写入。

若插入的数据主键已经存在,则会抛出 org.springframework.dao.DuplicateKeyException 异常,提示主键重复,不保存当前数据。

在 newdt 数据库中的 newcoll 集合中插入单条文档,示例代码如下:

```
use newdt
db.newcoll.insert({
    name:"jack",
```

```
    age:23,
    address:{
        city:"wuhan",
        code:434000,
        tel:123456
    }
})

#输出结果为
{ acknowledged: true,
  insertedIds: { '0': ObjectId("60658a16aa45a9378c309887") } }
```

在 newcoll 集合中插入多条文档，示例代码如下：

```
use newdt
db.newcoll.insert([{
    name:"tom",
    age:23,
    address:{
            city:"hangzhou",
            code:310000,
            tel:123456
    }
},{
    name:"marry",
    age:23,
    address:{
          city:"shanghai",
          code:200000,
          tel:123456
    }
}])

#输出结果为
{ acknowledged: true,
  insertedIds:
   { '0': ObjectId("60658efaaa45a9378c309888"),
     '1': ObjectId("60658efaaa45a9378c309889") } }
```

单击 MongoDB Compass 的刷新按钮，然后单击 newcoll 集合就可以看到刚才添加的集合了，如图 3-57 所示。

7. 更新文档

使用 updateOne()方法来更新单个文档，updateOne()方法在查询条件中即使查询到了满足该条件的多个文档，也只会更新第 1 个文档，它只用于更新单个文档。updateOne()方法的语法格式如下：

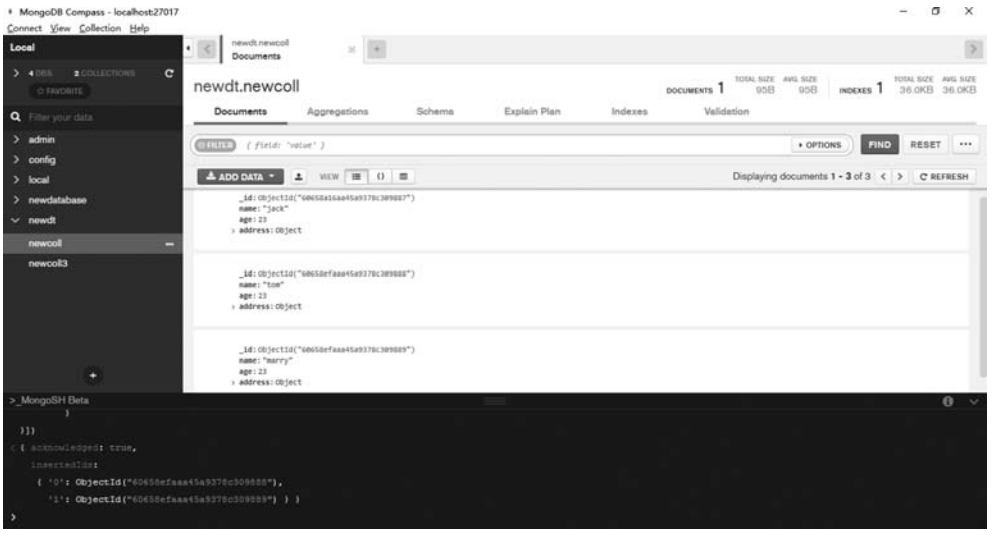

图 3-57　插入新的文档

```
db.collection.updateOne(
   <filter>,
   <update>,
   {
     upsert: <boolean>,
     writeConcern: <document>,
     collation: <document>,
     arrayFilters: [ <filterdocument1>, ... ],
     hint: <document|string> //Available starting in MongoDB 4.2.1
   }
)
```

filter 为查询条件，可以使用与 find()方法中相同的查询选择器，类似 SQL 的 WHERE 语句。update 为要应用的修改。upsert 为可选参数，如果设置为 true，则在没有文档符合查询条件时创建新的文档，默认值为 false。multi 为可选参数，用于更新满足条件的全部文档。writeConcern 为可选参数，表示写入策略，默认值为 1。collation 为可选参数，表示排序规则。arrayFilters 为可选参数，用于筛选文档的数组，确定要为数组字段上的更新操作修改哪些数组元素。hint 为可选参数，指定用于支持查询断言的索引的文档或者字符串。

在前文已经向 newcoll 集合插入了两个文档，对应于 name 字段分别为 tom 与 marry，现将 tom 的 city 修改为 shenzhen，示例代码如下：

```
use newdt
db.newcoll.updateOne(
{"name":"tom"},
{$set:{"address.city":"shenzhen"}}
)

#输出结果为
```

```
{ acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0 }
```

使用 updateMany()方法可更新所有满足条件的文档，updateMany()方法与 updateOne()方法格式和参数都一致，唯一不同的是即使 updateOne()方法查询到了多个符合查询条件的文档，也只会更新第 1 个，而 updateMany()方法则更新全部的文档。

前文向 newcoll 添加的所有文档，其 age 字段均为 23，以此为条件将 age 为 23 的所有文档中的 city 修改为 beijing，示例代码如下：

```
use newdt
db.newcoll.updateMany(
{"age":23},
{ $set:{"address.city":"beijing"}}
)

#输出结果为
{ acknowledged: true,
  insertedId: null,
  matchedCount: 3,
  modifiedCount: 3,
  upsertedCount: 0 }
```

注意 新版本的 MongoDB 已经弃用了 update()方法，但为了兼容性目前仍然可以使用。建议新开发的程序不要再使用 update()方法，更新单个文档使用 updateOne()方法，更新多个文档则应使用 updateMany()方法，而 save()方法也在 4.2 版本以后被弃用，可以使用 replaceOne()方法来代替。

8. 删除文档

使用 deleteOne()方法从数据集中删除一个文档，deleteOne()方法的语法格式如下：

```
db.collection.deleteOne(
   <filter>,
   {
      writeConcern: <document>,
      collation: <document>,
      hint: <document|string>            //Available starting in MongoDB 4.4
   }
```

filter 为删除条件。writeConcern 为可选参数，表示写入策略。collation 为可选参数，表示排序规则。hint 为可选参数，指定用于支持查询断言的索引的文档或者字符串。

使用 deleteOne()方法删除 newcoll 集合中 name 为 tom 的文档，代码如下：

```
use newdt
db.newcoll.deleteOne({"name":"tom"})

#输出结果为{ acknowledged: true, deletedCount: 1 }
```

使用 deleteMany()删除所有符合条件的文档,其语法格式如下:

```
db.collection.deleteMany(
   <filter>,
   {
      writeConcern: <document>,
      collation: <document>
   }
)
```

如果要删除数据集下的所有文档,则可以传递一个空的文档给 filter。示例代码为删除所有符合 age 等于 23 的文档,代码如下:

```
use newdt
db.newcoll.deleteMany({"age":23})

#输出结果为{ acknowledged: true, deletedCount: 2 }
```

注意 新版本的 MongoDB 已经弃用了 remove()方法,但为了兼容性目前仍然可以使用。建议新开发的程序不要再使用 remove()方法,删除单个文档使用 deleteOne()方法,删除多个文档则应使用 deleteMany()方法。

9. 查询文档

使用 find()方法查询,find()方法的语法格式如下:

```
db.collection.find(query,projection)
```

query 为可选参数,表示查询条件。projection 为可选参数,用于指定要在文档中返回的与查询条件匹配的字段。如要返回全部字段,则忽略此参数。

下面示例代码将三条文档插入 newcoll 数据集中,代码如下:

```
db.newcoll.insert([{
    name:"tom",
    age:23,
    address:{
            city:"hangzhou",
            code:310000,
            tel:123456
    }
},{
```

```
    name:"marry",
    age:23,
    address:{
            city:"shanghai",
            code:200000,
            tel:123456
    }
},{
    name:"jack",
    age:23,
    address:{
            city:"beijing",
            code:100000,
            tel:123456
    }
}
])

#输出结果为
{ acknowledged: true,
  insertedIds:
   { '0': ObjectId("6065adfeaa45a9378c30988b"),
     '1': ObjectId("6065adfeaa45a9378c30988c"),
     '2': ObjectId("6065adfeaa45a9378c30988d") } }
```

使用 find()方法将所有的文档查询出来，示例代码如下：

```
db.newcoll.find()

#输出结果为
{ _id: ObjectId("6065adfeaa45a9378c30988b"),
  name: 'tom',
  age: 23,
  ...
```

也可以使用条件参数将满足条件的文档查询出来，例如查询 city 为 beijing 的文档，代码如下：

```
db.newcoll.find({"address.city":"beijing"})

#输出结果为
{ _id: ObjectId("6065adfeaa45a9378c30988d"),
  name: 'jack',
  age: 23,
  address: { city: 'beijing', code: 100000, tel: 123456 } }
```

通过使用不同的查询条件，使查询变得灵活而强大，可以从不同的文档内查询到各种符合查询条件的数据，查询条件有多种匹配的运算符，具体的运算符如表 3-8 所示。

表 3-8 查询运算符

运算符	说　明
$ eq	等于
$ gt	大于
$ gte	大于或等于
$ in	匹配数组中的任意值
$ lt	小于
$ lte	小于或等于
$ ne	不等于
$ nin	不匹配数组中的任意值
$ and	与
$ not	取反
$ nor	或非
$ or	或
$ exists	匹配具有指定字段的文档
$ type	匹配指定类型字段
$ expr	聚合表达式
$ jsonSchema	根据给定的 JSON 模式验证文档
$ mod	对字段值执行模运算,并选择具有指定结果的文档
$ regex	选择与正则表达式匹配的文档
$ text	执行文本搜索
$ where	匹配满足 JavaScript 表达式的文档
$ geoIntersects	选择与几何图形相交的集合图形
$ geoWithin	选择便捷几何中的几何
$ near	返回点附近的地理空间对象
$ nearSphere	返回球体上某一点附近的地理空间对象
$ all	匹配包含查询中指定的所有元素的数组
$ elemMatch	匹配数组字段中元素匹配所有指定的 $ elemMatch 条件
$ size	匹配数组字段为指定大小
$ bitsAllClear	匹配数值或二进制值,其中一组比特位置的值都为 0
$ bitsAllSet	匹配数值或二进制值,其中一组比特位置的值都为 1
$ bitsAnyClear	匹配数值或二进制值,其中一组比特位置中的任意位的值都为 0
$ bitsAnySet	匹配数值或二进制值,其中一组比特位置中的任意位的值都为 1

常见运算符查询条件的示例代码如下：

```
# 查询满足 name 为 tom 或者 age 等于 23 的文档
db.newcoll.find({ $ or:[{"name":"tom"},{"age":23}]})

# 查询 age 小于 40 的文档
db.newcoll.find({"age":{ $ lt:40}})

# 查询 name 为 tom 同时 age 等于 23 的文档
db.newcoll.find({"name":"tom","age":23})
```

```
#查询存在 age 字段且 age 字段在 20～40 的文档
db.newcoll.find({"age":{ $exists:true, $nin:[20,40]}})
```

3.6 使用 Python 操作 MongoDB

3.6.1 MongoDB 操作模块

MongoDB 官方提供了 Python 操作 MongoDB 的驱动 PyMongo，MongoDB 官方也推荐使用 PyMongo 来对 MongoDB 进行操作。不同版本的 PyMongo 支持的 MongoDB 版本也不一样，如表 3-9 所示，此表为 PyMongo 对应支持的 MongoDB 版本。如果下载了旧版本的 PyMongo，对于新版本的 MongoDB 可能无法支持。

前文下载并安装的 MongoDB 版本为 4.4.4 版本，根据表 3-9 所示，要想对该版本的 MongoDB 进行正常操作，则需要安装 3.11 及以上的 PyMongo 版本才能正常使用。

表 3-9 不同版本的 PyMongo 对 MongoDB 的支持

Python Driver	Mongo DB 4.4	Mongo DB 4.2	Mongo DB 4.0	Mongo DB 3.6	Mongo DB 3.4	Mongo DB 3.2	Mongo DB 3.0	Mongo DB 2.6
3.11	√	√	√	√	√	√	√	√
3.1		√	√	√	√	√	√	√
3.9		√	√	√	√	√	√	√
3.8			√	√	√	√	√	√
3.7			√	√	√	√	√	√
3.6				√	√	√	√	√
3.5					√	√	√	√
3.4					√	√	√	√
3.3						√	√	√
3.2						√	√	√
3.1							√	√
3							√	√
2.9							√	√
2.8							√	√
2.7								√

使用 pip 可以很方便地安装 PyMongo，打开 PyCharm 编辑器，然后单击 View→Tool Windows→Terminal 打开终端窗口，在命令行中输入 pip install pymongo 并按回车键来安装 PyMongo 模块，等待提示 Successfully installed pymongo 后即可使用。

PyMongo 中使用 MongoClient()方法来连接 MongoDB 服务器，连接 MongoDB 服务器的语法格式如下：

```
MongoDB://[username:password@]host1[:port1][,...hostN[:portN]][/[defaultauthdb][?
options]]
```

连接 MongoDB 服务器并将该服务器上所有的数据库名列出来，其 Python 代码如下：

```
#第 3 章//mon.py
import pymongo                                  #导入 pymongo

#使用 MongoClient()方法连接 MongoDB 服务器
myclient = pymongo.MongoClient("mongodb://localhost:27017/")

#使用 list_database_names()方法获取所有数据库名
dblist = myclient.list_database_names()
print(dblist)

#输出结果为['admin', 'config', 'local', 'newdatabase', 'newdt']
```

3.6.2 使用 Python 操作 MongoDB

1. 创建数据库

在 MongoDB 的命令行模式中创建数据库，可在连接服务器后，使用“use 新数据库名”命令即可创建新的数据库，在 Python 中同样使用此方法来创建数据库，Python 创建 MongoDB 数据库的代码如下：

```
import pymongo
myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["pydatabase"]                   #创建一个名为 pydatabase 的数据库
```

当数据库中没有内容时，不会显示该数据库，所以此时使用 list_database_names()方法也是无法获取刚创建的数据库。可以在数据库内创建一个集合，并且插入一个文档，这样就可以看到新创建的数据库了，Python 中创建集合与创建数据库一样非常简单，创建集合的代码如下：

```
#第 3 章//mon.py
import pymongo
myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["pydatabase"]                   #创建一个名为 pydatabase 的数据库
mycoll = mydb["mycoll"]                         #创建一个名为 mycoll 的集合
doc = {"name":"jack","age":24}                  #定义一个文档
result = mycoll.insert_one(doc)                 #将文档插入集合
dblist = myclient.list_database_names()         #获取数据库列表
print(dblist)

#输出结果为['admin', 'config', 'local', 'newdatabase', 'newdt', 'pydatabase']
```

在上面的代码中，mydb＝myclient["pydatabase"]表示如果 pydatabase 数据库存在，则不创建新的数据库，如果 pydatabase 数据库不存在，则创建新的数据库并命名为 pydatabase。

2. 查看数据库

使用 list_database_names()方法来查看当前所有的数据库，list_database_names()方

法会返回一个列表，列表内为当前所有可以显示的数据库的名称，示例代码如下：

```
import pymongo
myclient = pymongo.MongoClient("mongodb://localhost:27017/")
dblist = myclient.list_database_names()                    #查看当前所有数据库
print(dblist)

#输出结果为['admin', 'config', 'local', 'newdatabase', 'newdt', 'pydatabase']
```

3. 删除数据库

使用 drop_database(name_or_database, session=None)方法可删除 MongoDB 数据库，删除数据库后，数据库内的数据也会一并删除，在实际操作中需谨慎使用该方法。该方法包含两个参数，name_or_database 参数为要删除的数据库实例，session 为会话。删除 pydatabase 数据库的示例代码如下：

```
#第 3 章//mon.py
import pymongo
myclient = pymongo.MongoClient("mongodb://localhost:27017/")
dblist = myclient.list_database_names()
print(dblist)
dropDatabase = myclient["pydatabase"]
myclient.drop_database(dropDatabase)                       #删除数据库
dblist = myclient.list_database_names()
print(dblist)

#输出结果为
['admin', 'config', 'local', 'newdatabase', 'newdt', 'pydatabase']
['admin', 'config', 'local', 'newdatabase', 'newdt']
```

4. 创建集合

使用 Python 创建集合与创建数据库一样非常简单，示例代码如下：

```
import pymongo
myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["pydatabase"]                              #创建一个名为 pydatabase 的数据库
mycoll = mydb["mycoll"]                                    #创建一个名为 mycoll 的集合
```

在上面的代码中使用数据库对象即可创建一个集合，mycoll=mydb["mycoll"]该段代码创建了一个名为 mycoll 的集合。

5. 查看集合

使用 list_collection_names()方法可查看当前数据库下所有的集合，返回值为列表(list)类型，列表中的值为集合的名称。示例代码如下：

```
import pymongo
myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["pydatabase"]          #pydatabase 数据库已存在，使用 pydatabase 数据库
```

```
listcoll = mydb.list_collection_names()          #查看当前数据库下所有的集合
print(listcoll)

#输出结果为['mycoll']
```

6. 删除集合

使用 drop()方法可删除当前集合,示例代码如下:

```
import pymongo
myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["pydatabase"]          #pydatabase 数据库已存在,使用 pydatabase 数据库
coll = mydb["mycoll"]
coll.drop()                            #删除当前集合
listcoll = mydb.list_collection_names()
print(listcoll)

#输出结果为[]
```

7. 插入文档

使用 insert_one()方法可插入单个文档,使用 insert_id 返回插入的_id 示例代码如下:

```
#第 3 章//mon.py
import pymongo
myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["pydatabase"]                          #使用 pydatabase 数据库
mycoll = mydb["mycoll"]                                #创建一个名为 mycoll 的集合
doc = {"name":"jack","age":24}                         #构造一个文档
result = mycoll.insert_one(doc)                        #插入单个文档
id = result.inserted_id                                #返回刚才插入文档的_id
print(id)

#输出结果为 6066838efd04788beb0f6b15
```

使用 insert_many()方法可插入多个文档,使用 inserted_ids 返回所有插入文档的 id 值,示例代码如下:

```
#第 3 章//mon.py
import pymongo
myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["pydatabase"]                          #使用 pydatabase 数据库
mycoll = mydb["mycoll"]                                #使用 mycoll 的集合
arraydoc = [
{"name":"jack","age":24},
{"name":"tom","age":25},
{"name":"marry","age":26},
{"name":"jan","age":27},
]
#构造文档列表
```

```
results = mycoll.insert_many(arraydoc)                    #插入多条文档
ids = results.inserted_ids                                #返回 id 列表
print(ids)

#输出结果为
[ObjectId('6066855cec0a22b64766563f'),
 ObjectId('6066855cec0a22b647665640'),
 ObjectId('6066855cec0a22b647665641'),
 ObjectId('6066855cec0a22b647665642')]
```

对于_id,除了可以使用系统生成的_id 字段以外,也可以插入指定_id 的文档。例如要插入一条指定_id 字段的文档,示例代码如下:

```
#第 3 章//mon.py
import pymongo
myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["pydatabase"]
mycoll = mydb["mycoll"]
doc = {"_id":12345,"name":"jack","age":24}                    #构造一个指定_id 的文档
result = mycoll.insert_one(doc)
id = result.inserted_id
print(id)

#输出结果为 12345
```

8. 更新文档

使用 update_one()方法更新单条文档,该方法常用两个参数,filter 与 update,filter 参数表示要更新文档的查询条件,update 表示要更新的文档,与 MongoDB 的 Shell 命令一致,update_one 即使查询出多条符合条件的文档,也只会更新第 1 条文档。示例代码如下:

```
#第 3 章//mon.py
import pymongo
myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["pydatabase"]
mycoll = mydb["mycoll"]
query = {"_id":12345}                                     #要更新文档的查询条件
doc = {"$set":{"name":"jack_update","age":99}}            #需要更新的字段
result = mycoll.update_one(query,doc)                     #更新一条文档
```

使用 update_many()方法可更新所有匹配条件的文档,该方法常用的两个参数为 filter 与 update,filter 参数表示要更新文档的查询条件,update 表示要更新的文档。示例代码如下:

```
#第 3 章//mon.py
import pymongo
myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["pydatabase"]
```

```
mycoll = mydb["mycoll"]
query = {"_id":12345}                                    #要更新文档的查询条件
doc = {"$set":{"name":"jack_update","age":99}}           #需要更新的字段
result = mycoll.update_many(query,doc)                   #更新所有满足查询条件的文档
```

9. 查询文档

使用 find_one()方法可查询集合中满足条件的一条文档,find_one()方法的主要参数为 filter,即查询条件,find_one()方法查询的条件即使有多条文档满足,也只会返回一条文档数据,其示例代码如下:

```
#第3章//mon.py
import pymongo
myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["pydatabase"]
mycoll = mydb["mycoll"]
query = {"_id":12345}                                    #查询条件
result = mycoll.find_one(query)                          #查询一条文档
print(result)

#输出结果为{'_id': 12345, 'name': 'jack_update', 'age': 99}
```

使用 find()方法可查询集合中所有满足条件的文档,find()方法的主要参数为 filter,即查询条件,使用 find()方法查询所有 age 小于 40 的文档,示例代码如下:

```
#第3章//mon.py
import pymongo
myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["pydatabase"]
mycoll = mydb["mycoll"]
query = {"age":{"$lt":40}}                               #查询所有 age 小于 40 的文档
result = mycoll.find(query)
for cur in result:
    print(cur)

#输出结果为
{'_id': ObjectId('6066818c74814628cc9a0...
```

匹配查询的运算符在前文 MongoDB 操作语法内已经讲解过,如有对匹配运算符不清楚的读者可以翻看前文。需要注意的是,在 MongoDB 的 Shell 中使用 find()方法其运算符是不需要加引号的,但是在 Python 中运算符需要使用引号括起来。find()方法查询的结果为 pymongo.cursor.Cursor,可以循环将字典对象也就是文档读取出来。

使用 limit()设置返回 find()查询的指定条数,该方法接收一个数字参数。如设置的指定条数大于 find()返回的总条数,则按 find()返回条数显示,示例代码如下:

```
#第3章//mon.py
import pymongo
myclient = pymongo.MongoClient("mongodb://localhost:27017/")
```

```
mydb = myclient["pydatabase"]
mycoll = mydb["mycoll"]
query = {"age":{"$lt":40}}                          #查询所有 age 小于 40 的文档
result = mycoll.find(query).limit(2)                #限制返回 2 条文档
for cur in result:
    print(cur)

#输出结果为
{'_id': ObjectId('6066818c74814628cc9a06c4'), 'name': 'jack', 'age': 24}
{'_id': ObjectId('606681a8a1d83301dfbfe9c9'), 'name': 'jack', 'age': 24}
```

10. 删除文档

使用 delete_one()方法可删除匹配条件的文档，该方法常用的参数为 filter，即匹配查询条件，delete_one()只会删除一条匹配的文档，即使该查询条件匹配出多条文档，也只删除一条文档。例如要删除 name 为 jack 的文档，示例代码如下：

```
#第 3 章//mon.py
import pymongo
myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["pydatabase"]
mycoll = mydb["mycoll"]
query = {"name":"jack"}                             #查询 name 为 jack 的文档
result = mycoll.delete_one(query)                   #删除一条匹配的文档
```

使用 delete_many()方法可删除所有匹配条件的文档，该方法常用的参数为 filter，即匹配查询条件，例如删除所有 age 小于 20 的文档，示例代码如下：

```
#第 3 章//mon.py
import pymongo
myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["pydatabase"]
mycoll = mydb["mycoll"]
query = {"age":{"$lt":"jack"}}                      #查询所有 age 小于 20 的文档
result = mycoll.delete_many(query)
```

如果要删除集合中内所有的文档，则传入的匹配条件为{}即可。

3.7 Redis 简介及安装

3.7.1 Redis 简介

Redis 是一个开源(BSD 许可)的内存中的数据结构存储系统，它可以用作数据库、缓存和消息中间件。它支持多种类型的数据结构，如字符串、散列、列表、集合、有序集合与范围查询、bitmaps、hyperloglogs 和地理空间(geospatial) 索引半径查询。Redis 内置了复制、LUA 脚本、LRU 驱动事件、事务和不同级别的磁盘持久化，并通过 Redis 哨兵和自动分区

提供高可用性。

Redis 执行的是纯内存操作，需要的时候可以持久化到硬盘中，且 Redis 是单线程，没有频繁的切换操作，数据结构简单，自己构建了 VM 机制，使用多路 I/O 复用模型，因为这些特性，所以 Redis 访问速度非常快。

由于 Redis 的访问速度快、高可用及支持数据类型丰富等特性，所以被大量地应用在缓存数据、消息队列、计数器、热点数据等请求量巨大且对实时性要求比较高的应用场景中。

3.7.2 Redis 安装

目前 Redis 官方已不再提供 Windows 平台下的 Redis 版本，且 Redis 官方推荐部署在 Linux 操作系统下，所以想要使用 Redis 则需要在 Linux 操作系统下进行安装。笔者使用的 Linux 版本为 CentOS 7 64 位操作系统。

首先打开 Redis 的官方网站，网址为 https://redis.io/，单击 Download 链接进入 Redis 下载页面，如图 3-58 所示。

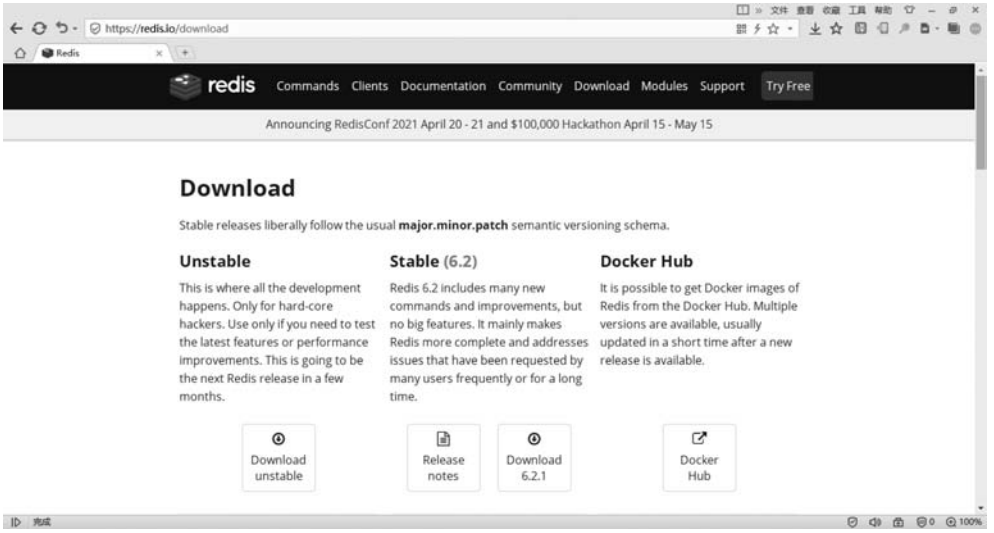

图 3-58 Redis 下载页面

在 Redis 下载页面有 3 个可选下载项，分别为 Unstable(不稳定版本)、Stable(稳定版)、Docker Hub(Docker 安装版)，在笔者写作期间 Redis 的最新版本为 6.2。

单击 Stable 版本的 Download 6.2.1 按钮下载 Redis 安装包，然后将下载好的 Redis 安装包上传到 CentOS 操作系统内。也可以在 Download 6.2.1 按钮上右击，在弹出菜单中选择“复制链接地址”将该安装包的下载链接复制到 CentOS 操作系统内进行下载，安装包的下载网址为 https://download.redis.io/releases/redis-6.2.1.tar.gz，进入 CentOS 操作系统进行下载，执行命令如下：

```
wget -i -c https://download.redis.io/releases/redis-6.2.1.tar.gz
```

下载完后可以得到一个 redis-6.2.1.tar.gz 的压缩包，将其解压出来的命令如下：

```
tar -zxf redis-6.2.1.tar.gz
```

解压完成后即可获得一个与压缩包同名的文件夹，也就是 redis-6.2.1，进入 redis-6.2.1 文件夹，执行命令如下：

```
cd redis-6.2.1
```

在进行 make 前，先确认是否安装了 gcc，如果没有安装 gcc 则编译会失败，失败时会提示 cc：command not found 错误。使用 yum 命令安装 gcc，命令如下：

```
yum install gcc
```

安装 gcc，等待出现 Complete 后执行 make 命令，命令如下：

```
make
```

有时会出现以下错误，如图 3-59 所示。

```
[root@localhost redis-6.2.1]# make
cd src && make all
make[1]: Entering directory `/root/redis-6.2.1/src'
    CC Makefile.dep
make[1]: Leaving directory `/root/redis-6.2.1/src'
make[1]: Entering directory `/root/redis-6.2.1/src'
    CC adlist.o
In file included from adlist.c:34:0:
zmalloc.h:50:31: fatal error: jemalloc/jemalloc.h: No such file or directory
 #include <jemalloc/jemalloc.h>
                               ^
compilation terminated.
make[1]: *** [adlist.o] Error 1
make[1]: Leaving directory `/root/redis-6.2.1/src'
make: *** [all] Error 2
```

图 3-59　编译 Redis 时出现的错误

当出现如图 3-59 所示的错误时需要在 make 后加上 MALLOC＝libc 强制对 libc malloc 进行编译，参数命令如下：

```
make MALLOC=libc
```

Redis 安装完毕后会提示 Hint：It's a good idea to run 'make test';，安装程序提示我们进行安装测试，输入命令如下：

```
make test
```

执行命令后提示 You need tcl 8.5 or newer in order to run the Redis test，使用 yum 安装 tcl，命令如下：

```
yum install tcl
```

安装完 tcl 后，再次执行 make test，命令如下：

```
make test
```

此时一切正常，表示 Redis 安装完成了，即使 make test 不通过也不必太过担心，并不影响 Redis 的使用。Redis 安装完成后会在 redis-6.2.1 目录下生成一个 src 文件夹，里面有 3 个可执行文件，分别为 redis-server、redis-benchmark 和 redis-cli，将这 3 个文件复制至/usr/redis 目录下，再将 redis-6.2.1 目录下的 redis.conf 也复制至/usr/redis 目录下，在 src 目录内执行的命令如下：

```
mkdir /usr/redis
cp redis-server /usr/redis
cp redis-benchmark /usr/redis
cp redis-cli /usr/redis
cd ..
cp redis.conf /usr/redis
```

此时在/usr/redis 目录下就有 4 个文件了，如图 3-60 所示。

```
[root@localhost redis]# ll
total 7116
-rwxr-xr-x. 1 root root  825240 Apr  3 23:36 redis-benchmark
-rwxr-xr-x. 1 root root  990800 Apr  3 23:36 redis-cli
-rw-r--r--. 1 root root   92222 Apr  3 23:36 redis.conf
-rwxr-xr-x. 1 root root 5371504 Apr  3 23:36 redis-server
[root@localhost redis]#
```

图 3-60　Redis 文件夹中的文件

进入/usr/redis 文件夹内，启动 Redis 的命令如下：

```
cd /usr/redis
./redis-server redis.conf
```

执行上面的命令后，Redis 就成功启动了，如图 3-61 所示。

在 Redis 运行界面中显示当前程序在独立模式下运行，端口号为 6379，PID 为 14019。此时不能将该界面关闭，当前属于前台运行方式，如果关闭了该界面则 Redis 就停止运行了，接下来将 Redis 设置为后台运行。

按快捷键 Ctrl+C 关闭当前运行界面，打开/usr/redis/redis.conf 配置文件，命令如下：

```
vi /usr/redis/redis.conf
```

在打开的 redis.conf 文件内按斜杠键/进行全文检索，在斜杠键后输入 daemonize，然后按回车键，此时文档会定位到 daemonize no 这一行，单击 i 键进行编辑，通过方向键控制光标的位置，将 daemonize 后的 no 修改为 yes，然后按 ESC 键，再键入：wq 进行保存，如图 3-62 所示。

此时再次启动 Redis，Redis 欢迎界面就不显示了，执行命令如下：

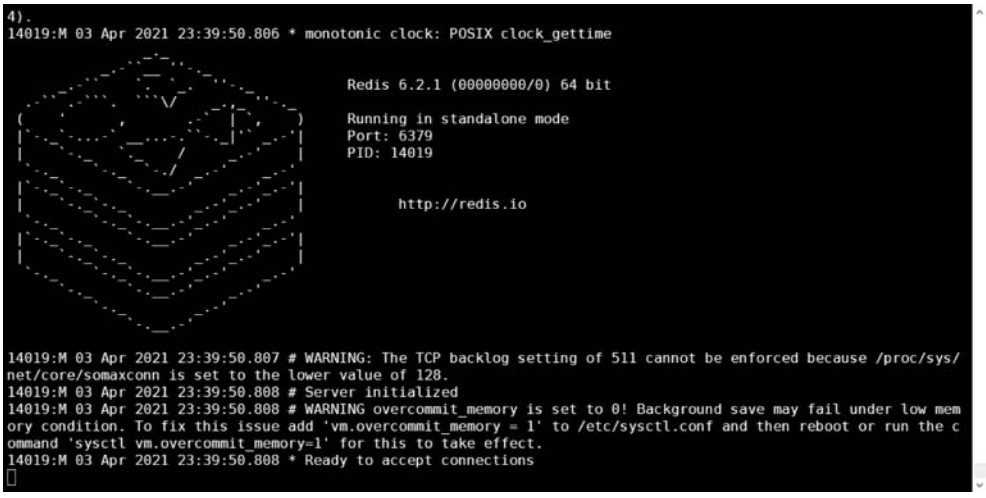

图 3-61　Redis 运行界面

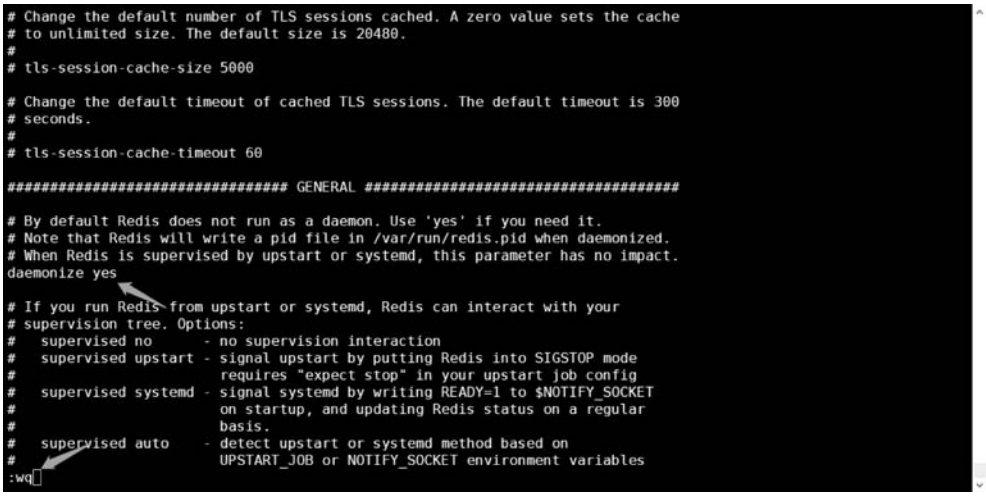

图 3-62　修改 Redis 配置文件

```
cd /usr/redis
./redis-server redis.conf
```

查看 Redis 是否运行，代码如下：

```
ps -ef | grep redis
```

此时 Redis 已经在后台运行了，如图 3-63 所示。

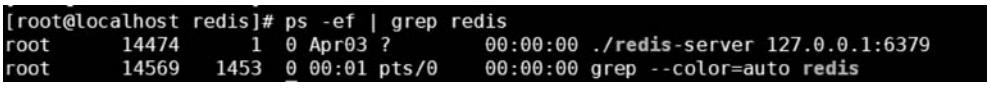

图 3-63　Redis 运行进程

3.7.3 Redis 可视化工具

不同于 MySQL 与 MongoDB，Redis 官方没有提供可视化工具，目前市场上比较流行的且比较完善的 Redis 可视化工具有 RedisDesktopManager 与 FastoRedis 等，这里主要讲解 RedisDesktopManager 工具的安装及使用。

RedisDesktopManager 是一款比较流行的 Redis 可视化工具，它起初是款开源产品，之后升级成为一款商业软件。其产品官网为 https://rdm.dev/。RedisDesktopManager 简称 RDM，适用于多种平台，是一个用于 Windows、Linux 和 macOS 的快速开源 Redis 数据库管理应用程序。该工具提供了一个易于使用的 GUI，可以访问 Redis 数据库并执行一些基本操作：将键视为树、CRUD 键、通过 Shell 执行命令。RDM 支持 SSL / TLS 加密。

打开 RedisDesktopManager 的官网 https://rdm.dev/，如图 3-64 所示。

图 3-64 RDM 官网

单击“价钱”链接进入产品试用界面，在 Windows 平台下有 3 个订阅计划，分别为个人、企业、Microsoft，单击“开始 14 天免费试用”按钮进行注册。输入你的邮箱及注册密码，单击“注册”按钮进行注册，此时会向你的邮箱发送一封邮件，如图 3-65 所示。单击 Confirm Your Email 按钮完成邮箱验证。

此时会跳转回 RDM 的官方网站，并会显示一个“开始免费的 14 天试用”的按钮，单击此按钮就可以进入下载页面进行下载了，如图 3-66 所示。下载 Windows 版本，单击 Windows 文字右侧的 2021.3.332 链接即可开始 RDM 的下载。

下载完 RDM 后双击安装程序便可进行安装，安装过程比较简单，此处略过。安装完毕 RDM 后，初次启动 RDM 需要输入邮箱及密码，如图 3-67 所示。

将刚才在 RDM 官网注册的邮箱及密码填入后，单击“登录”按钮即可来到 RDM 的主界面，如图 3-68 所示。此时的 RDM 没有内容，需要在连接 Redis 服务器后才可以进行下一步的操作。

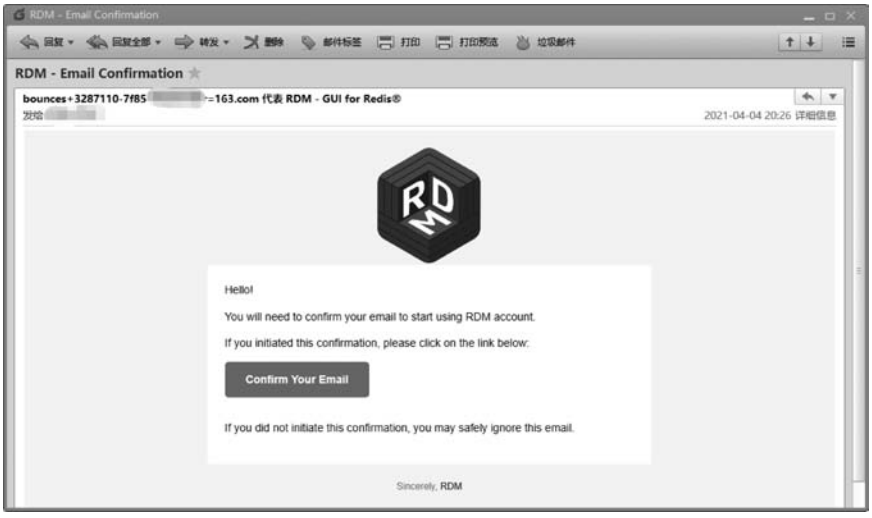

图 3-65　进行邮箱验证

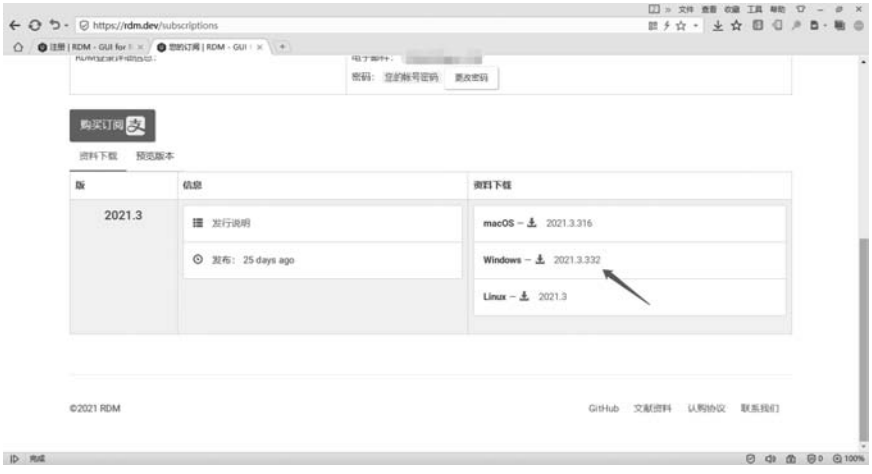

图 3-66　下载 RDM 测试版

图 3-67　初次打开 RDM 界面

图 3-68　RDM 主界面

使用 RMD 的第 1 步就是要连接 Redis 服务器，前面搭建好的 Redis 服务所在的服务器 IP 地址为 192.168.3.103，要连接该服务器此时需单击“连接到 Redis 服务器”按钮，这时会弹出一个窗口，需要对连接的服务器进行设置，如图 3-69 所示。

图 3-69　RDM 连接 Redis 服务器

如图 3-69 所示，需要填写远程服务器的相关参数才能进行连接，包含的参数如下。

名字：标识符，可以填写中文，用于区分 Redis 服务器。

地址：Redis 服务器的 IP 地址，因使用 SSH 的方式进行连接，所以此处填写 127.0.0.1，使用 SSH 方式连接相当于使用远程服务器的用户名及密码登录到远程服务器上对本机的 Redis 进行连接。

密码：因未设置 Redis 的密码，此处留空。

用户名：因未设置 Redis 的用户名，此处留空。

SSH 地址：远程服务器的 IP 地址。

SSH 用户：远程服务器登录的用户名。

密码：远程服务器登录的密码。

设置好以上参数后，单击“测试连接”按钮，此时 RDM 就可以连接上远程服务器的 Redis 服务了，然后单击“确定”按钮对 Redis 进行连接。此时在 RDM 的主界面会出现一个 Redis 的远程连接标识符，单击此标识符就可以看到远程 Redis 服务器上的所有数据库了，如图 3-70 所示。

图 3-70　远程 Redis 服务器的数据库列表

在 Redis 标识符的右侧，有一排操作按钮，分别是服务器信息、打开控制台、重载服务器、卸载所有数据、编辑连接设置、复制连接及删除连接。

单击“服务器信息”按钮，在 RDM 界面的右侧会显示当前连接的远程服务器的运行状态信息，如图 3-71 所示。在服务器信息界面，包含了全景信息、服务器信息、慢查询日志、客户端及推送/订阅通道等几个栏目。

全景信息：可以了解当前服务器上每秒查询率、连接的客户端数量、内存占用、网络输入、网络输出及键总量等内容，其内容都有图表的形式实时呈现，该界面可用于监测服务器的性能情况。

(1) 服务器：在该栏目下可以查看诸如 clients、cluster、command、cpu、errorstats、memory、persistence、replication、server、stats 等信息。

(2) 慢查询日志：在该栏目可以查看 Redis 中的慢查询日志。慢查询日志主要记录了

一些执行时间超过给定时长的 Redis 请求，让使用者更好地监视和找出在业务中的一些慢 Redis 操作，以便找到更好的优化方法。

（3）客户端：在该栏目下可以查看当前连接 Redis 的客户端 IP、时长及当前执行的库。

（4）推送/订阅通道：在该栏目下可以查看推送/订阅通道的情况。

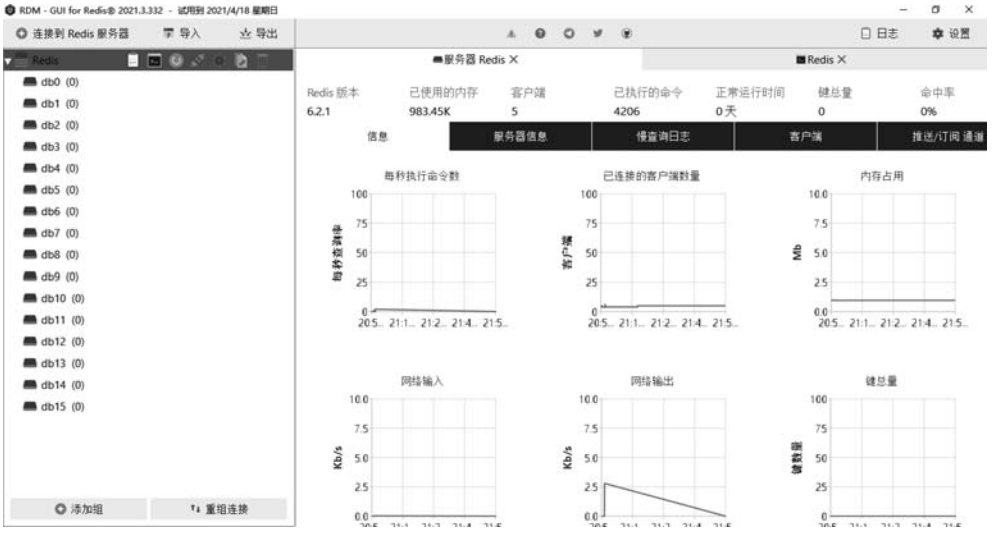

图 3-71　RDM 服务器信息界面

单击“打开控制台”按钮，在 RDM 界面的右侧会显示当前所连的 Redis Shell，如图 3-72 所示。可以在该窗口执行 Redis 的命令，如输入 ping，则 Redis 服务器会返回 PONG。

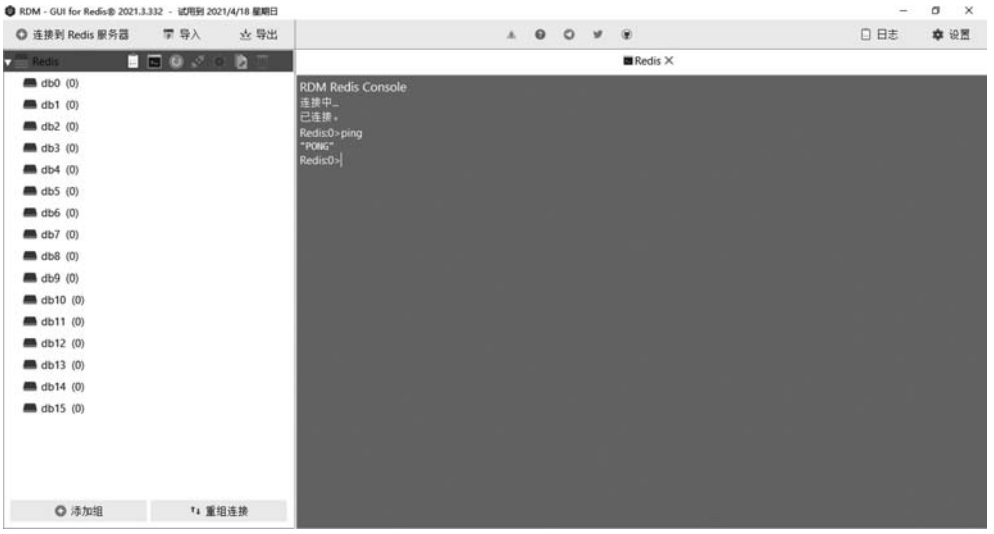

图 3-72　RDM 控制台

单击“重载服务器”按钮，会重新刷新服务器。单击“卸载所有数据”按钮，会清空所有数据。单击“编辑连接设置”按钮会打开连接配置对话框。单击“复制链接”按钮，会在 RDM 服务器列表里增加一个与当前连接一模一样的连接。单击“删除连接”按钮则会删除当前

连接。

单击 db0 数据库，在其右侧有个“添加新键”按钮，单击此按钮便可向 db0 中添加一个键名为 session1 的键，字段类型为 String，字段值为 12345，如图 3-73 所示。单击“保存”按钮，此时就完成了向 Redis 中 db0 数据库添加数据的操作。

图 3-73　RDM 添加字段

在 db0 下会多出一个 session1 的按钮，单击此按钮，在 RDM 右侧会看到键 session1 的值为 123456，如图 3-74 所示。在当前界面可以对键值进行重命名。

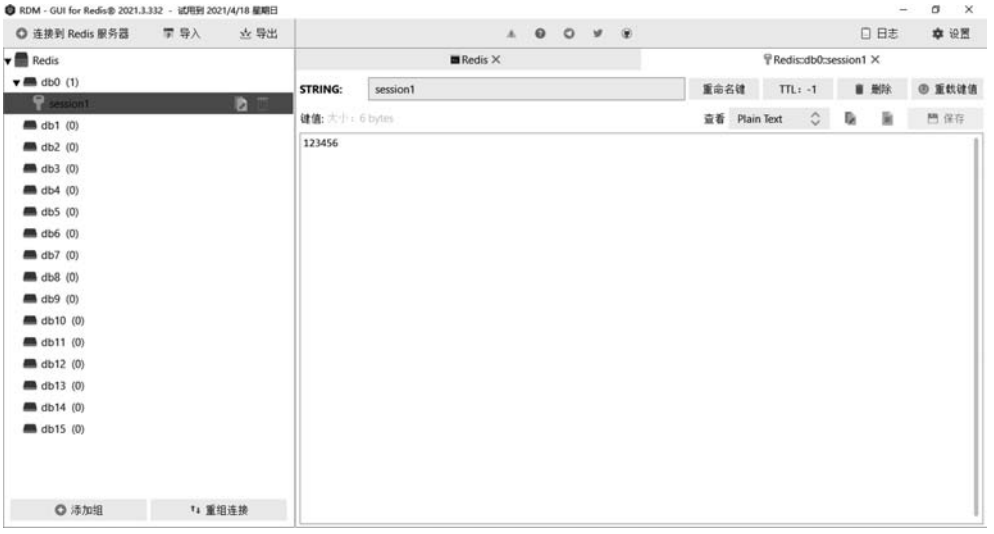

图 3-74　RDM 编辑字段

单击“重命名”按钮即可修改键值名称，在值处可以直接进行修改，修改完后右侧保存按钮便被激活。RDM 的整体界面比较简洁，操作也比较方便。

3.7.4 Redis 基础

Redis 是一个开源的高性能键值对(key-value)非关系型数据库,属于 NoSQL 产品类,因其速度快,数据类型丰富,常被用作缓存、消息队列、排行榜、电子商务秒杀等功能应用模块,与 MySQL 或者 MongoDB 进行互补。

Redis 6.2 支持 6 种数据类型,其数据类型包括 string、list、set、zset、hash、stream。其中 string 表示字符串类型。list 表示列表类型。set 表示集合类型。zset 表示有序集合类型。hash 表示哈希类型。stream 表示数据流类型。其中 stream 类型是 Redis 5.0 以后新增加的数据类型。其类型的具体说明如表 3-10 所示。

表 3-10 Redis 类型说明

数据类型	说 明
string(字符串)	可存储字符串、整数和浮点数
list(列表)	列表,它的每个节点都包含一个字符串
set(集合)	无序的集合,在此集合中的每个元素都是一个字符串,且不重复
zset(有序集合)	有序集合,可包含字符串、整数、浮点数、分值,排序大小由分值决定
hash(哈希散列)	键值对应的无序列表
stream(流)	消息链表,每条消息都有唯一的 ID 和对应的内容

在前面使用 RDM 时,细心的读者发现了当 RDM 连接到 Redis 后,立即显示了 16 个数据库,编号为 0～15,而之前使用 MySQL 与 MongoDB 时则不会出现这种情况。在 MySQL 与 MongoDB 中,数据库都是由用户自己创建的。

Redis 中创建的 16 个数据库的概念与传统的数据库其实是不太一样的,Redis 此时的数据库更像 MySQL 中表的概念,在 MySQL 的表中可以存放任何类型的字段和对应的值,而在 Redis 的 db 中,同样也存放着字段和值。与其说是数据库,其更像表,如图 3-75 所示。

图 3-75 Redis 数据库与 MySQL 表对比

至于为什么是 16 个数据库,这是由 Redis 的作者所设置的默认配置,此数量可以根据 Redis 配置文件进行修改,在 redis.conf 文件中有一行配置信息为 databases 16,如果将 16 修改为 1000 则 Redis 启动的时候就会自动创建 1000 个数据库了。默认创建多个数据库还有另一个原因是为了开发者更好地管理自己的数据,可以将不同功能的数据存放在不同的

数据库中以便维护。

Redis 不支持自定义数据库名字，所以每个数据库都以编号命名。开发者需要自己记录存储的数据与数据库的对应关系，且 Redis 不支持为不同的数据库设置不同的密码，客户端要么能访问全部的数据库，要么一个都访问不了。

3.8 Redis 操作语法

Redis 主要有 6 种类型的数据，而 Redis 的主要操作都是针对这 6 种数据类型进行的。以下所有的操作都是在 RDM 的控制台里进行的。

1. string(字符串)增、删、改、查

```
#增加一条键为 newkey,值为 str 的记录
set newkey "str"
ok

#查询 newkey 键的键值
get newkey
"str"

#将 newkey 键的键值修改为 newstr
set newkey "newstr"
ok

#将 newkey 键的名称修改为 nkey
rename newkey nkey
ok

#查找所有给定模式的 key
keys n*
"nkey"

#删除指定键
del nkey
"1"

#查询指定键是否存在
exists nkey
"0"

#删除当前数据库中的所有内容
flushdb
ok
```

2. set(集合)增、删、改、查

操作集合时需要注意的是，集合内的成员是不重复的、唯一的且成员类型是 string 类型。集合操作代码如下：

```
#创建一个集合并向其中添加 3 个成员
sadd setkey "123","abc","你好"
3

#查询指定集合里所有的成员
smembers setkey
"123"
"abc"
"你好"

#删除成员 abc,如果有则返回 1,如果没有则返回 0
srem setkey "abc"
1

#添加不重复成员,重复成员无法添加
sadd setkey "aaa"
1
```

3. list(列表)增、删、改、查

list 列表按照插入顺序排序,列表内容可以向列表的头部或者尾部添加元素,操作代码如下:

```
#创建一个 3 个元素的列表
lpush listkey "123","abc","你好"
3

#查询 listkey 的集合
lrange listkey 0 -1
"123"
"abc"
"你好"

#向 list 尾部添加元素
rpush listkey "new"
6

#向 list 头部添加元素
lpush listkey "lnew"
7

#更新 index 为 1 的值
lset listkey 1 "index1"
ok

#删除 index 为 1 的值
lrem listkey 1 "index1"
1
```

4. hash(哈希)增、删、改、查

hash 是一个 string 类型的 field 和 value 的映射表,适合存储对象,操作代码如下:

```
#创建一个 hash 集合 hashtable,并将 key 设置为 user,将 value 设置为 jack
hset hashtable "user" "jack"
1

#获取 hash 中字段的数量
hlen hashtable
1

#获取 hash 中所有的 key
hkeys hashtable
"user"

#返回 hash 中所有的 value
hvals hashtable
"jack"

#向 hash 添加记录
hset hashtable "pass" "123456"
1

#获取指定键的值
hget hashtable user
"jack"

#获取当前 hash 所有的键与值
hgetall hashtable
"user"
"jack"
"pass"
"123456"

#更新 hashtable 中 user 的值
hset hashtable user "123"
0

hgetall hashtable
"user"
"123"
"pass"
"123456"
```

5. zset(有序集合)增、删、改、查

zset(有序集合)与 set(集合)一样,其元素为 string 类型,且不允许重复成员的出现。与 set 不同的是 zset 每个元素都会关联一个 score(分数),Redis 通过分数来为集合中的成员进行排序。有序集合的成员是唯一的,但 score 允许相同,操作代码如下:

```
#创建一个有序集合并向其中添加一个成员,且将分数设为1
zadd zsetkey 1 "str"
"1"

#向有序集合内再添加一个成员,且将分数设为2
zadd zsetkey 2 "sec"
"1"

#查询有序集合内所有的值
zrange zsetkey 0 -1
1) "str"
2) "sec"

#删除有序集合内的成员
zrem zsetkey "str"
1

#显示有序集合内的所有成员
zrange zsetkey 0 -1
1) "str"
```

3.9 使用 Python 操作 Redis

3.9.1 Redis 操作模块

Redis 官方没有提供 Python 语言的操作模块,但是市场上有大量由第三方开发的 Python 模块,Redis 官方也将这些第三方模块收集并公示在其官方网站上了。Redis 的官方网站为 https://redis.io,打开其官方网站,单击 Clients 链接即可进入 Redis 官方收集的针对各种语言的客户端,如图 3-76 所示。

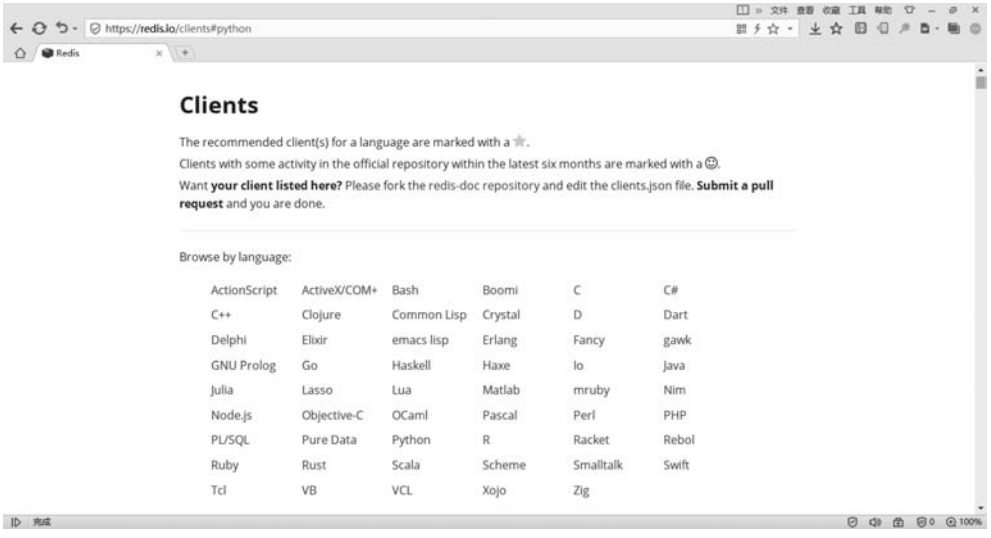

图 3-76 其他语言对 Redis 的支持

在该页中 Redis 官方提示，在其推荐的客户端后标有星星图标。此处单击 Python 链接便可跳转到 Python 客户端的列表中，如图 3-77 所示。

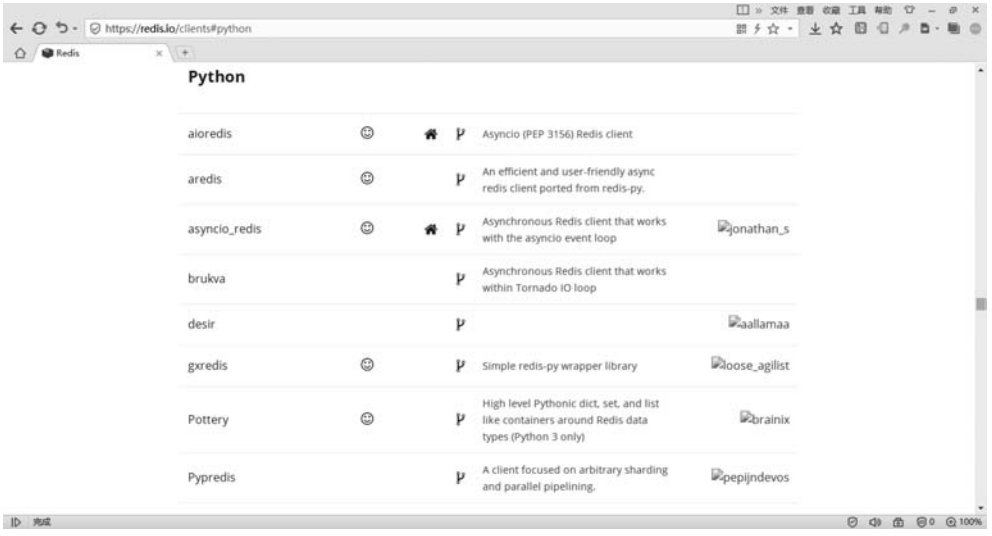

图 3-77 支持 Redis 的 Python 客户端列表

在前面 Redis 官方提示过，推荐使用的客户端后标有黄色小星星，往下滑动 Python 语言的客户端列表，发现有两个 Python 客户端被 Redis 官方推荐，分别为 redis-py 与 walrus，如图 3-78 所示。

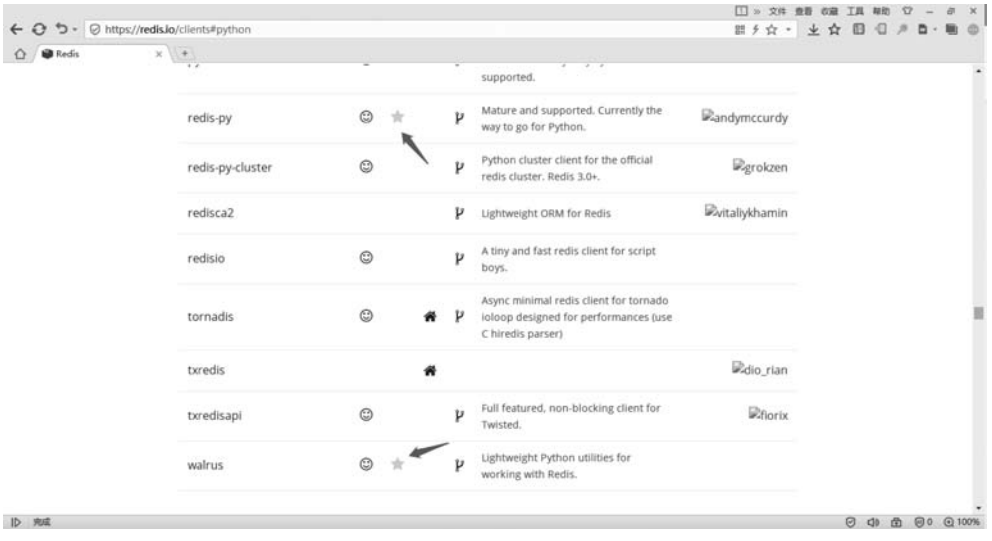

图 3-78 Redis 官方推荐的 Python 客户端

对于这两个库 Redis 官方对其分别描述为，redis-py 成熟且有支撑，符合目前 Python 的发展方向。walrus 为轻量级 Python 库。根据以上信息，选择 redis-py 用作 Python 对 Redis 的开发会更加合适。

单击 redis-py 黄色小星星右侧的分支图标即可进入 redis-py 的 GitHub 项目，其链接地

址为 https://github.com/andymccurdy/redis-py，如图 3-79 所示。

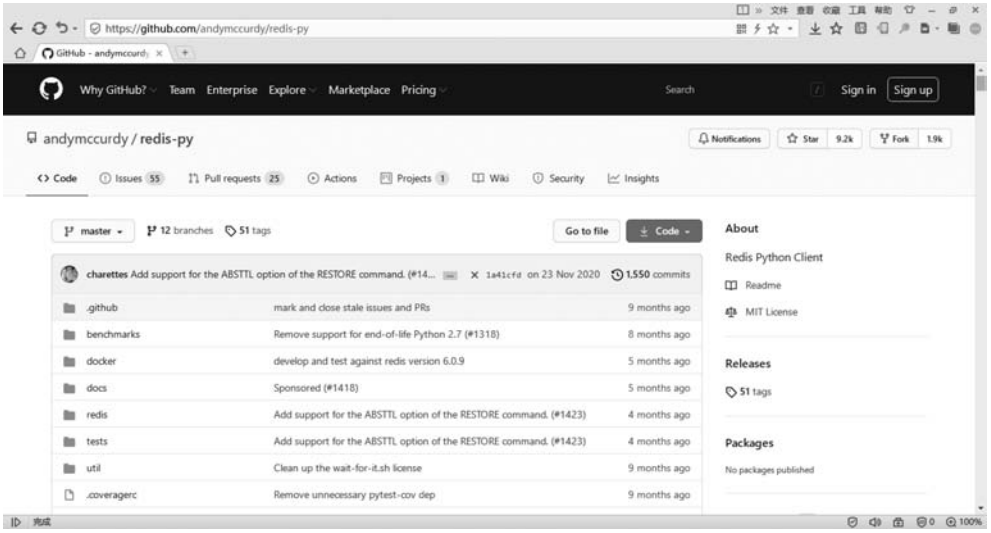

图 3-79　redis-py 项目的 GitHub 页面

在 redis-py 项目的说明中提到，redis-py 3.5.x 将是支持 Python 2 的最后一个版本，在 2020 年 8 月 1 日之前将继续为 Python 2 修复错误及提供安全补丁，在这之后将停止对 Python 2 的支持，而 redis-py 4.0 将是未来的主要版本，且要使用 redis-py 4.0 需要 Python 3.5 以上的版本。

根据其提示目前 redis-py 3.5.x 为最新版且仍然支持 Python 2，但仅提供错误修复及安全补丁，不排除下一个版本可能就不再支持 Python 2。在 Python 3 上是可以正常使用的。redis-py 可以通过 pip 进行安装，打开 PyCharm 并进入终端，使用 PyCharm 终端安装 redis-py，其安装命令如下：

```
pip install redis
```

出现 Successfully installed redis-3.5.3 字样即表示安装成功，如图 3-80 所示。

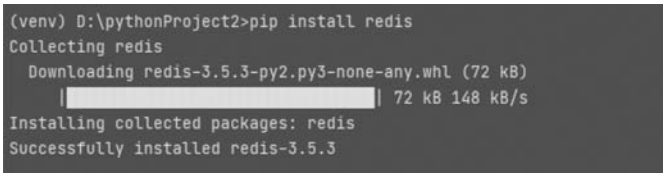

图 3-80　安装 redis-py

3.9.2　使用 Python 操作 Redis

redis-py 项目的 GitHub 页上提到了 redis 模块提供了两个类，分别是 Redis 和 StrictRedis，redis-py 3.0 不再支持传统的 Redis 客户端类，StrictRedis 已被重命名为 Redis 并且提供了一个名为 StrictRedis 的别名，以便以前使用 StrictRedis 的用户可以继续运行而

不改变代码。redis-py 不支持集群模式。

1. 连接 Redis

如果使用普通连接方式,则需要注意的是应在 CentOS 操作系统上的防火墙将 Redis 服务器端口设置对外开放,临时开放命令如下:

```
iptables -I INPUT 1 -p tcp -m state --state NEW -m tcp --dport 6379 -j ACCEPT
```

且将 redis.conf 中的 bind 127.0.0.1 -::1 替换为 bind 0.0.0.0 -::1,将 protected-mode 设置为 no,否则无法连接 Redis。

redis 模块下的 Redis 类构造函数拥有丰富的参数,可以进行多种方式的连接,例如普通连接、SSH 连接等,其参数如表 3-11 所示。

表 3-11 Redis 构造函数参数

参 数	说 明
host	连接的主机地址
port	连接 Redis 服务的端口
db	默认连接 Redis 数据库
password	Redis 的密码
socket_timeout	socket 超时时间
socket_connect_timeout	socket 连接超时时间
socket_keepalive	socket 的长连接
socket_keepalive_options	socket 长连接设置
connection_pool	连接池
unix_socket_path	socket path
encoding	编码
encoding_errors	错误处理方案
charset	字符集
decode_responses	返回结果是否为 decode
retry_on_timeout	重试超时时间
ssl	ssl 连接方式
ssl_keyfile	ssl keyfile
ssl_certfile	ssl certfile
ssl_cert_reqs	设置 ssl 安全检查模式
ssl_ca_certs	证书路径
max_connections	最大连接数
single_connection_client	单用户连接限制
health_check_interval	心跳检测
client_name	客户端名称
username	用户名

使用 Redis 进行普通连接 Redis 服务器的代码如下:

```
import redis
r = redis.Redis(host = '192.168.3.106', port = 6379, db = 0)
```

```
r.set('foo', 'bar')
print(r.get('foo'))

#输出结果为 b'bar'
```

如需登录跳板机连接 Redis 服务器，且使用跳板机的用户名和密码登录，需要引入 sshtunnel 模块，虽然 Redis 类提供了 SSH 的连接方式，但是无法使用跳板机的用户名和密码进行验证。首先安装 sshtunnel 模块，pip 命令如下：

```
pip install redis
```

出现 Successfully 即表示安装成功，sshtunnel 模块中有个 SSHTunnelForwarder 类，该类用于初始化到远程服务器的 SSH 隧道，其构造函数常用的参数包括 ssh_address_or_host（SSH 地址）、ssh_username（SSH 连接的用户名）、ssh_password（SSH 连接的密码）及 remote_bind_address（远程机器地址和端口号），使用 SSH 连接 Redis 服务器的具体代码如下：

```
#第 3 章//rds.py
import sshtunnel
import redis
server = sshtunnel.SSHTunnelForwarder(
    ssh_address_or_host = "192.168.3.106",                  #远程服务器地址
    ssh_username = "root",                                  #远程服务器登录用户
    ssh_password = "root",                                  #远程服务器登录密码
    remote_bind_address = ('192.168.3.106',6379)            #远程服务器 IP 及端口号
)
server.start()
conn = redis.Redis(
    host = '127.0.0.1',                                     #Redis 所在服务器地址
    port = server.local_bind_port,                          #Redis 端口
    decode_responses = True                                 #返回方式为字符串
)
print("已连接")

#输出结果为已连接
```

对于新手读者来讲，有个常见的错误需要注意一下，有种情况在使用 redis 模块时会提示 partially initialized module 'redis' has no attribute 'Redis'，如图 3-81 所示。

```
  File "D:\pythonProject2\redis.py", line 3, in <module>
    conn=redis.Redis()
AttributeError: partially initialized module 'redis' has no attribute 'Redis' (most likely due to a circular import)
```

图 3-81　使用 Redis 模块报错

这里需要注意了，一是检查 import redis 是否有拼写错误，二是看一看当前文件名或者同项目下是不是有文件名取了 redis.py 的名字。

通常在实际使用 Redis 连接时更偏向于使用连接池进行连接和管理，如果采取直连的

方式，使用 Redis 时要进行连接，不用时又释放了连接，而频繁地连接和释放是比较浪费资源的，所以通常情况下使用连接池的方式进行管理连接，连接池的原理是通过预先创建的多个连接，当进行 Redis 操作时，直接获取已经创建的连接进行操作，操作完成后不会释放连接，将用于后续其他的 Redis 操作，这样就达到了避免频繁地对 Redis 进行连接和释放的目的，从而提高了性能。redis 模块采用 ConnectionPool 来管理对 Redis Server 的所有连接。连接池连接 Redis 的代码如下：

```
import redis
pool = redis.ConnectionPool(host = 'localhost', port = 6379,db = 0)
red = redis.Redis(connection_pool = pool)
red.set('key1', 'value1')
red.set('key2', 'value2')
```

在 SSH 模式下使用连接池，代码如下：

```
#第 3 章//rds.py
import redis
import sshtunnel
server = sshtunnel.SSHTunnelForwarder(
    ssh_address_or_host = "192.168.3.106",
    ssh_username = "root",
    ssh_password = "root",
    remote_bind_address = ('192.168.3.106',6379)
)
server.start()
pool = redis.ConnectionPool(host = '127.0.0.1', port = server.local_bind_port,db = 0)
red = redis.Redis(connection_pool = pool)
red.set('key1', 'value1')
red.set('key2', 'value2')
```

2. string(字符串)增、删、改、查

```
#第 3 章//rds.py
import redis
pool = redis.ConnectionPool(host = 'localhost', port = 6379,db = 0)
red = redis.Redis(connection_pool = pool)

#增加一个 key 为 key1,value 为 value1 的值
red.set('key1', 'value1')

#查询 key1 的值
red.get("key1")

#修改 key1 的值
red.set('key1', 'value2')

#将 key1 的键名修改为 key2
```

```
red.rename("key1","key2")

#获取所有的键名
red.keys()

#删除 key2
red.delete("key2")

#查询 key2 是否存在
red.exists("key2")

#清空当前数据库中所有的 key
red.flushdb()
```

3. set(集合)增、删、改、查

```
#第 3 章//rds.py
import redis
pool = redis.ConnectionPool(host = 'localhost', port = 6379,db = 0)
red = redis.Redis(connection_pool = pool)

#创建一个集合并向其中添加 3 个成员
red.sadd("setkey","123","abc","你好")

#查询指定集合里所有的成员
red.smembers("setkey")

#删除成员 abc,如果有则返回 1,如果没有则返回 0
red.srem("setkey","abc")

#添加不重复的成员,重复的成员无法添加
red.sadd("setkey","aabbcc")
```

4. zset(有序集合)增、删、改、查

```
#第 3 章//rds.py
import redis
pool = redis.ConnectionPool(host = 'localhost', port = 6379,db = 0)
red = redis.Redis(connection_pool = pool)

#创建一个有序集合并向其中添加 2 个成员,且将分数分别设为 1 和 2
mapping = {"str":1,"sec":2}
red.zadd("zsetkey",mapping)

#向有序集合内再添加一个成员,且将分数设为 2
red.zadd("zsetkey",{"thr":2})

#查询有序集合内所有的值
res = red.zrange("zsetkey",0, - 1,withscores = True)                    #withscores 带上分数
```

```
for i in res:
    print(i)
#输出结果为
(b'str', 1.0)
(b'sec', 2.0)
(b'thr', 2.0)

#删除有序集合内的成员
red.zrem("zsetkey","str")
```

5. list(列表)增、删、改、查

```
#第 3 章//rds.py
import redis
pool = redis.ConnectionPool(host = 'localhost', port = 6379,db = 0)
red = redis.Redis(connection_pool = pool)

#创建一个含 3 个元素的列表
red.lpush("listkey","123","abc","你好")

#查询 listkey 的集合
res = red.lrange("listkey",0, -1)
for i in res:
    print(i)
#输出结果为
b'\xe4\xbd\xa0\xe5\xa5\xbd'
b'abc'
b'123'

#向 list 尾部添加元素
red.rpush("listkey","new")

#向 list 头部添加元素
red.lpush("listkey","lnew")

#更新 index 为 1 的值
red.lset("listkey",1,"index1")

#删除 index 为 1 的值
red.lrem("listkey",1,"index1")
```

6. hash(哈希)增、删、改、查

```
#第 3 章//rds.py
import redis
pool = redis.ConnectionPool(host = 'localhost', port = 6379,db = 0)
red = redis.Redis(connection_pool = pool)
```

```
#创建一个 hash 集合 hashtable,并将 key 设置为 user,将 value 设置为 jack
red.hset("hashtable","user","jack")

#获取 hash 中字段的数量
red.hlen("hashtable")

#获取 hash 中所有的 key
red.hkeys("hashtable")

#返回 hash 中所有的 value
red.hvals("hashtable")

#向 hash 添加记录
red.hset("hashtable","pass","123456")

#获取指定键的值
print(red.hget("hashtable","user"))

#获取当前 hash 所有的键与值
red.hgetall("hashtable")
#输出结果为{b'user': b'jack', b'pass': b'123456'}

#更新 hashtable 中 user 的值
red.hset("hashtable","user","123")
red.hgetall("hashtable")
#输出结果为{b'user': b'123', b'pass': b'123456'}
```

不论是 MySQL、MongoDB 还是 Redis,Python 程序能在其之上良好地运行得益于对数据库本身的特点特性、语法结构及使用方法的掌握,理解了数据库本身的原理、逻辑及使用方法,则立即就可以使用 Python 对其进行操作,这些 Python 模块都是根据数据库自身的方法进行构造的,使用起来非常简单和方便。