第 5 章

MySQL 数据库中的权限体系

提到权限，通常都是用户 A 拥有对象 B 的权限，很多朋友想必已经对此形成了思维定势，毕竟像 Oracle 或 SQL Server 这类大型数据库软件中的权限验证，也都是如此设定，指定甲用户拥有操作乙对象的权限。

而 MySQL 数据库的权限验证在设计阶段就体现的有所不同，它在这中间又加了一级维度，变成从丙处来的那个甲拥有访问乙的权限。可能有些同学一下子转不过弯来，那我换一个角度来描述：甲只有从丙处连接过来，才能够访问对象乙。这样对比的话，是否又跟 Oracle/SQL Server 这类数据库的身份验证机制比较相似了呢，只是如 Oracle 这类数据库软件，默认是不加丙这一层的（如果想加当然也可以支持），而在 MySQL 中，丙（来源）成了一个必选项，也就是说，对于 MySQL 数据库，甲的身份当然重要，但甲从哪儿来的也同样重要，即使同样叫“甲”，从 A 处来和从 B 处来的甲的权限可以不同，甚至应该视作是两个不同的用户。

在本章正式开始前先描述这样一段，并不是想说 MySQL 有多么高级或先进，只是想表达这样一种看法，MySQL 确实有所不同。OK，接下来，跟随三思一起，进入 MySQL 的权限世界吧！

5.1 谈谈权限处理逻辑

所有权限认证的根本目的，都是为了让用户只能做允许它做的事情，MySQL 也不例外，大家（泛指数据库产品）实现的原理也都差不多，只不过机制上稍有差异，在权限粒度控制上有所不同。

MySQL 数据库服务采用的是白名单的权限策略，也就是说，明确指定了哪些用户能够做什么，但没法明确地指定某些用户不能做什么，对权限的验证主要是通过 mysql 库下的几个数据字典表，来实现不同粒度的权限需求，关于这几个字典表后面会有章节详细介绍。这里简要介绍其处理逻辑，MySQL 在检查用户连接时可以分为两个阶段。

5.1.1 能不能连接

当用户发出请求尝试连接 MySQL 服务时，MySQL 首先是检查登录用户的相关信息，比如发起登录请求的主机名是否匹配、登录使用的用户名或密码是否正确，如果这一关过不去，那连接就直接被拒绝了，常见的登录失败信息“ERROR 1045 (28000): Access denied

for user '...'”，就是在这个阶段的验证未通过抛出的错误提示。

MySQL 数据库验证权限有 3 个维度：我是谁、从哪儿来、到哪儿去（真像哲学家探讨人生的终极命题呀）。这 3 个维度中，前两个决定能不能连接，就是说验证用户的身份是否合法，本步通过之后，才会涉及能不能访问目标对象的环节。

在 MySQL 数据库中验证用户，需要检查 3 项值：用户名、用户密码和来源主机，这 3 项信息的正确值（创建用户时指定），保存在 mysql 库中的 user 表对象内，分别对应 user 表对象中的 user、password 和 host 三列。如果事先看过 user 表中这几列的定义，会发现 MySQL 的设计非常有意思，这 3 列居然都可以为空（注意不是 NULL 值），这也是某些场景里登录 MySQL 数据库不需要输入用户名或不需要输入密码的原因。

5.1.2　能不能执行操作

连接到数据库之后，能不能执行操作，比如说建库、建表、改表，查询或修改数据等，这个阶段涉及的因素（对象）要复杂一点点，除了上面提到的 mysql.user 字典表起作用外，另外还有 mysql.db、mysql.tables_priv、mysql.columns_priv、mysql.proc_priv（事实上在 5.6.10 版本以前，还有 mysql.host 表，不过之后版本中，host 表已经明确被废弃，其实在之前版本里它也没什么用，原本就是被判了死缓，现在缓期过完了，不过没有转为无期，而是直接执行死刑）几个字典表来对数据库，或针对对象甚至是对象列做更细粒度的控制。

这些字典表虽说各有分工，但相互之间在权限分配上还是会有一定的重合，比如说 tables_priv 字典表一看就知道是专门针对表对象的权限明细，不过 user 表和 db 表中也可以授予用户操作表对象的权限。那么 MySQL 服务是怎么来区分这些权限的呢？我的个人理解，总的原则仍然是按照粒度。

比如要执行对整个数据库服务的管理操作，那么一定是根据 user 表中的记录验证权限是否匹配，因为只有这个表是针对 MySQL 服务全局的；如果请求某个明确的数据库对象，比如更新某个表中记录，那么 MySQL 服务也仍然会按照粒度从粗到细的方式，先检查 user 字典表中全局的设置，找不到匹配的话，则继续检查 db 字典表这样的方式；一旦在某个粒度匹配到合适的权限，就允许用户执行，否则继续查询更细的粒度表；如果所有的粒度滤过一遍，还是没能匹配到合适的权限，那么用户的操作就会被拒绝了。

通过上述逻辑还可以明确一点，就是粒度控制越细，权限验证上的步骤就会越多，相应对性能必然会有影响，这一点在进行权限分配时务必考虑在内。

5.1.3　权限变更何时生效

向用户分配的权限，哪些情况下会生效呢？一般来说，MySQL 数据库在启动时就会将前面提到的几个权限字典表中的内容读到内存里，当有用户连接或执行操作时，根据内存中的数据来检查用户是否有权限执行相应的操作。

注意，如果你读的足够认真并且大脑持续在进行思考，这会儿应该会产生这样的一个

疑问：如果用户连接上数据库后，管理员对该用户的权限进行了修改操作，是否即时生效呢？针对这个问题，答案是：看情况！

- 如果是通过 GRANT、REVOKE、SET PASSWORD、RENAME USER 等 MySQL 提供的命令执行修改，那么权限将马上生效，因为这些命令将触发系统重新载入授权表（GRANT TABLES）到内存。
- 如果是手动修改字典表方式（INSERT、UPDATE、DELETE），没错，MySQL 中可以手动修改字典表中的记录达到变更用户权限的目的，但这种情况下权限并不会马上生效，除非重启 MySQL 服务，或者 DBA 主动触发授权表的重新装载。

问题又来了，授权表被重新加载后，对当前已连接的客户端又会产生哪些影响呢？具体如下：

- 表或列粒度的权限将在客户端下次执行操作时生效。
- 数据库级的权限将在客户端执行 USE db_name 语句，切换数据库时生效。
- 全局权限和密码修改，对当前已连接的客户端无效，下次连接时才会生效。

5.2　权限授予与回收

当前 MySQL 就剩 system 一个系统管理员账户了，完全不符合业务需求啊，怎么办呢，本节就来着重演示 MySQL 数据库中如何创建用户、分配权限及回收权限。

在 MySQL 数据库里对于用户权限的授予和解除比较灵活，既可以通过专用命令，也可以通过直接操作字典表来实现，正所谓条条道路通目标。不过话说回来，修的马路多不叫奇迹，何况在这片神奇的土地，奇迹这个词本身就是奇迹，因此三思真是不好意思用奇迹这样的词来形容：这样想象不到的不平凡的事（注：该段描述为现代汉语词典中关于奇迹一词的解释），因此，我决定用一种加强的语气来描述我的感受：

比奇迹更神奇的是，这条条大路居然都修成了高速路。

比神奇的奇迹更神奇的是，这些高速路居然都是免费的。

比神奇的神奇奇迹更神奇，那就是神迹啊，额地神哪，免费的高速路居然也不堵车，这肯定不是二环、三环和四环，当然跟 G6/G8 线应该也没啥关系，至少也是十八环外了，弟兄们，走吧，跟着三思去溜达溜达~~~

再次提示

很多 Linux/UNIX 下管理 MySQL 数据库服务的 DBA，初看到数据库的管理账户 root 就发蒙了，以为这是什么重要的征兆，其实是大可不必的。此 root 非彼 root，MySQL 数据库里的 root 账户跟操作系统中的 root 没有丝毫的关联，只是数据库初始化时自动创建的一个名称而已。在本书第 3 章初始化数据库时，三思已经手动将该用户更名为了 system，我们的操作能够成功，并且未对后续数据库的正常管理带来任何异常，也说明 root 这个账户名不具备什么特殊的含义，完全可以随意处理。

基于合适的用户做符合其权限的事的目的，执行与权限相关操作的用户当然也得有权限，默认我们使用的是系统管理员账户，就是 system 用户了，本例中所做的用户管理操作，如非特别注明，均是使用 mysql 中的 system 用户执行。

5.2.1　创建用户

在创建用户之前，首先说明两点：

- 用户名的长度不能超过 16 个字符。
- 用户名和密码对大小写敏感，也就是说，Jss 和 jss 是两个不同的用户，密码也是如此。

1．传统方式创建

MySQL 中专用的创建用户的命令是 CREATE USER，该命令语法如下：

```
CREATE USER user_specification
    [, user_specification] ...

user_specification:
    user
    [
        IDENTIFIED BY [PASSWORD] 'password'
      | IDENTIFIED WITH auth_plugin [AS 'auth_string']
    ]
```

CREATE USER 命令是最传统的创建用户方式，语法看起来还是挺简单的，不过事实上与用户权限相关的细节非常有讲究，因为简单，所以灵活，因为灵活，所以可配置性强，因为可配置性强，所以细节很重要。

不过，刚开始接触时，大家倒是不用关注太多，从易到难嘛，咱们先按照最简单的方式创建一个名为 jss 的用户吧，执行操作如下：

```
(system@localhost) [mysql]> create user jss;
Query OK, 0 rows affected (0.01 sec)
```

你猜怎么着，成功了！不要担心“0 rows affected”那个提示，对于操作用户这类 SQL 语句，它的返回就是这个样子，只要不是返回什么 ERROR 之类提示，就是成功了，如果想看到明确的结果，可以通过查询 mysql.user 字典表中的记录验证一下：

```
(system@localhost) [mysql]> select user,host,password from mysql.user where user='jss';
+------+------+----------+
| user | host | password |
+------+------+----------+
| jss  | %    |          |
+------+------+----------+
1 row in set (0.00 sec)
```

当然啦！最好的验证方式仍然是登录测试，我们刚刚创建的用户，既没有设置登录的密码，也没有指定来源主机，因此该用户可以从任意安装了 MySQL 客户端，并能够访问

目标服务器的机器上创建连接。

换台装有 MySQL 客户端的服务器登录试试，例如：

```
[mysql@mysqldb02 ~]$ mysql -ujss -h 192.168.30.243
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 8
Server version: 5.6.12-log JSS for mysqltest

Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

(jss@192.168.30.243) [(none)]>
```

可以看到当前就是以 jss 身份连接到 30.243 服务器。由于前面创建用户时并没有指定任何密码，因此连接时无需指定密码即可顺利登录数据库。

2．修改用户密码

想必读者朋友也都看出来了，这样登录很不安全，密码可以有。那么怎么给用户设置密码呢？ALTER USER？NONONO，我们一般都不会这样干，甚至在 **MySQL 5.6.6** 版本之前，根本就没有提供 ALTER USER 这样的语法。“怎么会这样”，您是否在心里暗自问自己这个问题，其实若对 MySQL 的用户与权限体系有全面的认识，就会明白这种设计，对于 MySQL 数据库来说是合乎逻辑的。

MySQL 数据库中的用户没有太多属性，从前面的 CREATE USER 语法就能看得出来，与用户相关的选项，除了必须指定的用户名外，就是一个密码选项（唯一一个选项居然还不是必选项）。至于用户权限的授予，则是由单独的 SQL 命令操作（后面会介绍这些命令）。因此对于用户来说，可能变更的就是用户的密码，针对这一点需求，MySQL 没必要整出一个 ALTER USER 语法，它只需要单独针对修改密码的操作，提供一条命令即可，于是就有了 SET PASSWORD 命令，该命令语法如下：

```
SET PASSWORD [FOR user] =
    {
        PASSWORD('some password')
      | OLD_PASSWORD('some password')
      | 'encrypted password'
    }
```

比如，修改 jss 用户的密码为 5ienet.com，执行命令如下：

```
(jss@192.168.30.243) [(none)]> set password for jss=password('5ienet.com');
Query OK, 0 rows affected (0.00 sec)
```

SET PASSWORD 命令会自动更新系统授权表，之后再使用 jss 用户连接 MySQL 数据库，就必须输入密码才行，否则就会抛出：

```
ERROR 1045 (28000): Access denied for user 'jss'@'192.168.30.203' (using password: NO)
```

说一下 SET PASSWORD 命令中各选项的功能：

（1）SET PASSWORD：固定的语法格式，照着抄即可。

（2）[FOR user]：FOR 选项用于指定要修改密码的用户，如果是修改当前用户的密码，可以不用指定这个选项，如果要修改其他用户（前提是操作者确实有权限），那么必须通过 FOR 选项指定要修改的目标用户，格式为 user@host。

（3）PASSWORD/OLD_PASSWORD：这是两个密码专用函数。MySQL 数据库中用户密码当然不会是以明文的形式保存，它可不像国内某些专业 IT 社区那样，打着专业旗号却干出很不专业的事情。MySQL 中能够查询到的用户密码是按照它自己的加密逻辑处理后的字符串形式。在修改密码时，也必须指定加密后的字符形式保存，否则登录验证就会碰到异常。可是，都说了是加密后的形式，那我们又怎么能知道字符被加密后是什么形式呢？这里就要分两点来看：

①第一种是用户确实知道，甭管它是通过什么方式获得的（确实有多种方式），那么在指定密码时就可以直接指定其加密后的形式。

②第二种是用户不知道加密后的字符是什么，那么就可以由 MySQL 来帮助我们生成，MySQL 数据库提供了相应的函数 PASSWORD()，直接调用该函数即可，这种方式是最常见的调用方式，我们前面的示例中也是采用这种方式。

提示：关于 OLD_PASSWORD()函数

这个函数的命名容易产生误解，看起来仿佛是跟用户的旧密码有什么关系，其实不是这样，它只是为了应对 MySQL 的版本兼容性才出现的。在 4.1 之前的版本中，PASSWORD()函数生成 16 位长度的加密字符串，而在之后的版本中，为了提高安全性，MySQL 改进了密码的生成算法，现在生成的为 41 位长度的加密字符串，那么就会出现一个兼容性方面的问题，当用户使用 4.1 之前的客户端连接 MySQL 服务时，就会出现由于加密格式不统一造成的登录失败。为了提高兼容性，MySQL 新增加了 OLD_PASSWORD()函数，仍然采用原始的加密策略生成 16 位长度的加密字符串，管理员在设置用户口令时，就可以使用这个函数生成密码，使其能够兼容 4.1 之前版本的 MySQL 客户端。

两个函数处理相同字符串的输出如下：

```
(jss@192.168.30.243) [(none)]> select password('123456'),old_password('123456');
+-------------------------------------------+--------------------------+
| password('123456')                        | old_password('123456') |
+-------------------------------------------+--------------------------+
| *6BB4837EB74329105EE4568DDA7DC67ED2CA2AD9 | 565491d704013245       |
+-------------------------------------------+--------------------------+
```

前面提到，在 5.6.6 版本之前，MySQL 数据库都没有 ALTER USER 语法，那么为什么后来又增加了 ALTER USER，这个语法又能用来做什么呢？为什么增加这个语句我也没想明白，不过这个语句的功能可能要让很多人打死都猜不到。新增的 ALTER USER 语句的功能，与其他数据库软件中的 ALTER USER 功能差异巨大，一言以蔽之，就是让用户的密码

过期。注意一定要正确理解，是密码过期，而不是用户过期哟。用户仍然可以用（登录），只是密码过期后，无法做任何操作。

比如说，我们先将 jss 用户密码设置为过期，执行操作如下：

```
(system@localhost) [mysql]> alter user jss password expire;
Query OK, 0 rows affected (0.00 sec)
```

而后再以 jss 用户登录，用原始密码仍然能够登录成功，但是执行操作就不行喽：

```
(jss@192.168.30.243) [(none)]> show databases;
ERROR 1820 (HY000): You must SET PASSWORD before executing this statement
```

实践过之后，您是否回忆起了什么，或者说您现在应该知道，第 2 章 RPM 包方式安装后连接数据库，必须先修改用户密码才能执行操作，是如何实现的了吧！

3．通过登录主机验证用户

话说 MySQL 数据库中，用户登录除了验证用户名和密码外，不是号称还要检查来源主机呢嘛，怎么前面的登录操作，似乎并未感到有对主机层的验证呢？这个嘛，因为创建用户时就没有指定登录主机啊，没指定，默认就是不限制。不过这个“不限制”指的是不做限制，实际上字典表中还是会有对应的标识，查询一下 mysql.user 字典表中的信息：

```
(system@localhost) [(mysql)]> select user,host from mysql.user where user='jss';
+------+------+
| user | host |
+------+------+
| jss  | %    |
+------+------+
1 rows in set (0.00 sec)
```

注意到这条记录中 host 列的值了没，显示一个%（百分号）。熟悉 SQL 语法的朋友都知道，%在 SQL 语法中是作为通配符，代表任意字符串，在这里出现则代表任意主机，这个才是前面所说的不限制登录主机的真正原因。

没错，主机名可以指定通配符，规则与标准的 SQL 语法中定义完全相同：

- %：对应任意长度的任意字符。
- _：对应一位长度的任意字符。

如果 user 字典表中的 host 列值为空或%，均代表任意主机。因此，如果希望创建的用户只能从某个主机或某个 IP 段访问，那么在创建用户时，就必须明确指定 host，指定的 host 既可以是 IP，也可以是主机名，或者是可正确解析至 IP 地址的其他自定义名称。

接下来我们尝试创建一个名为 jss_ip 的用户，并且该用户仅允许从 192.168.30.203 的主机连接至 MySQL 服务端，执行命令如下：

```
(system@localhost) [(mysql)]> create user jss_ip@'192.168.30.203' identified by 'jss';
Query OK, 0 rows affected (0.00 sec)
```

这样使用 jss_ip 用户登录时，只有从 192.168.30.203 主机发出登录请求才能成功，从非 192.168.30.203 的主机上，使用 jss_ip 用户连接时，不管密码是否正确，都会抛出“ERROR 1045 (28000): Access denied”错误信息：

```
$ mysql -ujss_ip -pjss -h 192.168.30.243
```


```
ERROR 1045 (28000): Access denied for user 'jss_ip'@'192.168.10.113' (using password: YES)
```

如果希望 192.168.30.%网段的主机均能够使用 jss_ip 用户连接，又该如何设置呢？这种情况下就该通配符出马了：

```
(system@localhost) [(none)]> create user jss_ip@'192.168.30.%' identified by 'jss';
Query OK, 0 rows affected (0.00 sec)
```

而后从 192.168.30.%网段的任意主机上尝试连接 MySQL 服务器，都能够顺利登录：

```
$ mysql -ujss_ip -pjss -h 192.168.30.243
Welcome to the MySQL monitor.  Commands end with ; or \g.
```

其他大型数据库软件，直接指定用户即可登录数据库，但在 MySQL 数据库中，则额外还需要有主机这一维度，用户和主机（'user'@'host'）组成一个唯一**账户**，登录 MySQL 数据库时，实际上是**通过账户**进行验证。

由于 host 能够支持通配符，使得登录验证时来源主机的部分更加灵活，表 5-1 列举了一些 user 和 host 的常见组合，希望能够有助于大家理解。

表 5-1　用户与主机组合示例

user 列	host 列	对应连接情况
'jss'	'192.168.1.2'	使用 jss 用户登录时，只有从 192.168.1.2 主机发出登录请求才能成功创建连接
'jss'	'www.5ienet.%'	使用 jss 用户登录时，可以从主机名为 www.5ienet.(net/com/cn....)的任意主机创建连接
'jss'	'www.5ienet.com'	使用 jss 用户登录时，只能从主机名为 www.5ienet.com 的主机发出请求才能成功创建连接
'jss'	'%'	可以从任意主机使用 jss 用户连接
''	'10.0.0.%'	可以从 10.0.0.%网段内的任意主机创建连接，并且无需输入任何用户信息
''	'%'	任意主机均可以创建连接，并且连接过程中无需用户信息

大家是否注意到表 5-1 中前几行记录中的用户名都叫 jss，不过实际上它们不仅不是同一条记录，甚至不是一个用户。因为 MySQL 数据库是根据'user'@'host'来确认记录是否唯一，user 表中每一条记录都是一个独立的账户，每一个独立的账户都可以拥有各自的权限设置。

这种设计对于初接触 MySQL 数据库的朋友的确可能带来困扰，因为大家一般都只听过有 user，谁能想到这中间还夹着一层 host，不过我举个例子大家应该就明白了。比如说您有两位同事，都叫杨伟（user），一个从山东（host）来，另一个从山西（host）来，您就知道他们肯定不是一个人，这种情况搁现实生活中叫**重名**，两个确实是各自独立的个体。

“重名”说尽管能够帮助大家理解 user+host 的组合，不过朋友们可能还是会有疑问，就是重名所带来的现实尴尬，比方说有可能碰到你喊一声“美女”，结果一堆人答应的场景，那 MySQL 数据库中会不会出现这种情况呢？它又怎么保证一定是那个你想搭讪的姑娘回应呢？按照我的理解，拿这个问题拷问 MySQL 的智商实在太难为它了，别说 MySQL 搞

不清楚，就是换个活生生的人也搞不定啊，因此，肯定的答复就是，MySQL 保证不了。

不过放心啦，MySQL 不会返回一堆记录让人无所适从的，因为规矩是限定死的嘛，只能有一条回应，当然啦，它也不会随随便便挑一个给你。作为一款数据库软件，“严谨”是烙印在它的基因中的，MySQL 遇到这种情况，会按照既定的规则来处理，处理的规则归根结底就两个字：排序，而后从排好序的结果中取第一条记录。

MySQL 在排序时会将最明确的 host 值放在前面，比如说某个具体的主机名或 IP 地址就非常明确，而像通配符“%”就是最不明确的代表（它代表任意主机），排序时会放在后面，空字符串"尽管也表示任意主机，但排序的优先级比'%'更低，它会放在最后。对于 host 相同的记录，MySQL 会再按照 user 列中的值排序，规则与 host 完全相同，都是最明确的值放在最前面。

举例来说，user 字典表中有下列的记录：

```
+-----------+----------+-
| Host      | User     | ...
+-----------+----------+-
| %         | system   | ...
| %         | jss      | ...
| localhost | system   | ...
| localhost |          | ...
+-----------+----------+-
```

按照 MySQL 数据库的规则，排序好之后的结果类似这样：

```
+-----------+----------+-
| Host      | User     | ...
+-----------+----------+-
| localhost | system   | ...
| localhost |          | ...
| %         | jss      | ...
| %         | system   | ...
+-----------+----------+-
```

提 示

排序是什么时候做的呢？要知道，MySQL 在服务启动时就会将 user 表读取到内存中，在读取的过程中就会排序。MySQL 服务运行过程中，修改用户权限触发权限更新时，会刷新内存中的字典表，这期间又会进行排序，也就是内存中的字典表永远都是排好序的。

客户端创建连接时使用的用户名和主机，有可能同时匹配 user 表中的多条记录。在上面给出的例子中，使用 system 用户登录就有可能既匹配 system@'localhost'，又匹配 system@'%'两条记录。按照前面介绍的规则，如果是在 localhost 本地执行登录，那么一定会匹配为 system@'localhost'这个用户，否则的话，则会是 system@'%'这个用户了。

再给一个例子，user 表中有以下两条记录：

```
+---------------+----------+-
| Host          | User     | ...
```


```
+----------------+----------+-
| www.5ienet.com |          | ...
| %              | jss      | ...
+----------------+----------+-
```

当用户使用 jss 用户并且从 www.5ienet.com 主机登录 MySQL 数据库时，会匹配第一条记录，如果是从其他主机登录的话则是匹配第二条记录。实际上，从 www.5ienet.com 主机登录 MySQL 的话，是否指定用户根本就**没有区别**，因为 www.5ienet.com 已经非常明确，并且 user 列值为空字串，也就代表着只要是从 www.5ienet.com 主机发出的登录请求，不管指定的用户是什么（甚至可以是 user 表中不存在的用户），均会匹配为这条记录。

4．GRANT 方式创建用户

CREATE USER 只是创建用户的高速路之一，如果你觉得这条道路实在太过平坦，路边风景太过平淡，行程太过平常，不妨在抵达目的地之前，拐弯开上 GRANT 大道，饱览不一样的风景。

GRANT 命令并非本小节重点，这里仅简要描述一下其语句中与用户相关的部分：

```
GRANT priv_clause TO user [IDENTIFIED BY [PASSWORD] 'password'] ...
```

与创建用户相关的语法，看起来跟 CREATE USER 是差不多的嘛，事实上当然不是差不多，根本就是一模一样嘛，下面举个例子，操作如下：

```
(system@localhost) [(mysql)]> grant select on jssdb.* to jss_grant@192.168.30.203 identified by 'jss';
Query OK, 0 rows affected (0.00 sec)

(system@localhost) [(mysql)]> select user,host,password from mysql.user where user ='jss_grant';
+-----------+----------------+-------------------------------------------+
| user      | host           | password                                  |
+-----------+----------------+-------------------------------------------+
| jss_grant | 192.168.30.203 | *284578888014774CC4EF4C5C292F694CEDBB5457 |
+-----------+----------------+-------------------------------------------+
1 row in set (0.00 sec)
```

上述语句在实现了前面第 3 个例子（创建用户 jss_ip）的功能外，还额外授予了 jss_grant 用户查询 mysql.user 表的权限。MySQL 的开发团队靠着永不屈服、永不放弃、永不退缩、永不言败的奋争精神，用智慧和巧妙的构思完美复制了 ORACLE GRANT 语句的功能，这是全世界默默无闻的 MySQL 开发人员长期以来内生品格的自然流露，是全世界默默无闻的 MySQL 开发人员开拓前进的不竭动力，这就是传说中的“瑞典梦”。

5．另类方式创建用户

如果说觉得上述方式都不顺手，或者，大脑短路导致短暂忘记了命令的语法，那也没关系，mysql.user 表还记得吧，直接向该字典表中插入记录（一般 INSERT 语法想忘不容易），也是靠谱的，例如：

```
(system@localhost) [(none)]> insert into mysql.user (host,user,password,ssl_cipher,x509_issuer,
x509_subject) values ('192.168.30.203','jss_insert',password('jss'),'','','');
Query OK, 1 row affected (0.00 sec)
```


```
(system@localhost) [(none)]> select user,host,password from mysql.user where user ='jss_insert';
+------------+---------------+-------------------------------------------+
| user       | host          | password                                  |
+------------+---------------+-------------------------------------------+
| jss_insert | 192.168.30.203 | *2845788880147774CC4EF4C5C292F694CEDBB5457 |
+------------+---------------+-------------------------------------------+
1 row in set (0.00 sec)
```

手动修改权限字典表后，需要执行 FLUSH PRIVILEGES 语句，重新加载授权信息到内存中，否则手动修改的权限不会生效，执行操作如下：

```
(system@localhost) [none]> flush privileges;
Query OK, 0 rows affected (0.00 sec)
```

接下来可以尝试从 192.168.30.203 主机，分别使用 jss_insert 和 jss_ip 登录，对比看看效果，不仅看起来相同，实际表现也是一模一样。

这点跟 Oracle 数据库就截然不同了，Oracle 这类数据库是绝对不建议用户修改数据字典表的，而且一般情况下也不知道都应该改**哪些**地方（没错，完全可能不止一处需要修改），因此对于 Oracle 数据库，最安全、最稳妥也最快捷的方式，还是老老实实按照 Oracle 提供的命令进行操作。而 MySQL 则完全不同，官方不仅完全不介意用户通过操作字典表的方式进行功能修改（想想也是，连软件都是开源的，在这种地方设什么障碍也没有意义），甚至鼓励通过这种方式。话说回来，截止到 MySQL5.6.12 版本，都还没有提供修改“用户属性”的 ALTER USER 的语法，因此如果想对用户属性做修改，直接 UPDATE mysql.user 表就算是比较便捷的方式了。

当然啦，MySQL 中的用户其实也没什么属性可供修改，大多都是权限，唯一称得上属性又有修改可能的，就是用户的密码信息了。前面介绍过 SET PASSWORD 语句，用于修改用户密码非常专业，但并不是唯一的方法，在 MySQL 数据库中，我们可以使用更加直接的方式。实际上之前就这么干过，还记得第 3 章中修改 root 用户密码时所做的操作吗？没错，用户的信息保存在 mysql.user 字典表中，我们直接修改该表也是一样的。

例如，直接修改字典表，将 jss 用户的密码变更为 123456，执行操作如下：

```
(system@localhost) [(none)]> update mysql.user set password=password('123456') where user='jss' and
host='%';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

5.2.2　授予权限

用户管理的核心就是权限分配，MySQL 数据库中授予权限有专用命令 GRANT，它不仅能够授予权限，甚至还能创建用户（前面小节中演示过）。严谨些描述，它能在创建用户的同时授予权限，看起来授权操作倒像是顺带的功能一样。

GRANT 命令的语法看起来可是相当复杂的呐：

```
GRANT
    priv_type [(column_list)]
```


```
    [, priv_type [(column_list)]] ...
  ON [object_type] priv_level
  TO user [IDENTIFIED BY [PASSWORD] 'password'] ...
  [REQUIRE {NONE | ssl_option [[AND] ssl_option] ...}]
  [WITH with_option ...]
```

除了 priv_type，其他几个加粗的子项详细语法如下：

- object_type:

```
  TABLE
| FUNCTION
| PROCEDURE
```

- priv_level:

```
  *
| *.*
| db_name.*
| db_name.tbl_name
| tbl_name
| db_name.routine_name
```

- ssl_option:

```
  SSL
| X509
| CIPHER 'cipher'
| ISSUER 'issuer'
| SUBJECT 'subject'
```

- with_option:

```
  GRANT OPTION
| MAX_QUERIES_PER_HOUR count
| MAX_UPDATES_PER_HOUR count
| MAX_CONNECTIONS_PER_HOUR count
| MAX_USER_CONNECTIONS count
```

貌似漏掉了 priv_type 选项，放心我没忘，最重要的 priv_type 需要放在最显著的地方解说。它看起来最简单，但可选项也最多，用于指定可授予（或收回）的权限类型，对此官方文档中，针对可授予的权限，专门列了个表罗列的很清晰（表 5-2）。

表 5-2　MySQL 用户权限

权限类型	简要说明
ALL [PRIVILEGES]	授予除 GRANT OPTION 外的所有权限
ALTER	允许执行 ALTER TABLE 操作
ALTER ROUTINE	允许修改或删除存储过程和函数
CREATE	允许创建数据库和创建表对象
CREATE ROUTINE	允许创建存储过程和函数
CREATE TABLESPACE	允许创建、修改或删除表空间及日志文件组
CREATE TEMPORARY TABLES	允许执行 CREATE TEMPORARY TABLE 语句创建临时表

续表

权限类型	简要说明
CREATE USER	允许执行 CREATE USER、DROP USER、RENAME USER 和 REVOKE ALL PRIVILEGES 语句
CREATE VIEW	允许创建/修改视图
DELETE	允许执行 DELETE 语句
DROP	允许删除数据库/表或视图
EVENT	允许使用 Event 对象
EXECUTE	允许用户执行存储程序
FILE	允许用户读写文件
GRANT OPTION	允许将授予的权限再由该用户授予其他用户
INDEX	允许创建/删除索引
INSERT	允许执行 INSERT 语句
LOCK TABLES	允许对拥有 SELECT 权限的表对象执行 LOCK TABLES
PROCESS	允许用户执行 SHOW PROCESSLIST 命令查看当前所有连接
PROXY	允许使用 PROXY
REFERENCES	尚未应用
RELOAD	允许执行 FLUSH 操作
REPLICATION CLIENT	允许用户连接复制环境中的 Master/Slave
REPLICATION SLAVE	允许复制环境的 Slave 端从 Master 端读取数据
SELECT	允许执行 SELECT 语句
SHOW DATABASES	允许执行 SHOW DATABASES 语句显示所有数据库
SHOW VIEW	允许执行 SHOW CREATE VIEW 查看视图定义
SHUTDOWN	允许通过 mysqladmin 命令关闭数据库
SUPER	允许执行管理操作，比如 CHANGE MASTER TO、KILL、PURGE BINARY LOGS、SET GLOBAL 等语句
TRIGGER	允许创建或删除触发器
UPDATE	允许执行 UPDATE 操作
USAGE	意指没有权限（no privileges）

这个表罗列了所有可授予用户的权限，不管针对什么用户，授予哪个对象，授予什么粒度的权限，都是从表 5-2 中的关键字中选择。

以上几段加一块基本上就是 GRANT 语句的语法，看起来呢是复杂了一点点，不过不懂也没关系，再说就算懂了也不一定记得住，就算记住了也不一定真能理解它在说什么。

就像现在人人都知道要先感谢国家，你懂的，人人都明白那不过就是说说（不过海外各种二代及二代亲戚们说这话时应该是真心的），关键时刻得动真格的，得会用才行，三思争取后面多弄几个例子，让大家伙都搞明白这个事儿。

下面先举个最简单的例子帮助大家理解，我们要授予 jss_grant@'192.168.30.203'用户查询 mysql.user 表的权限，执行语句如下：

```
(system@localhost) [(none)]> grant select on mysql.user to jss_grant@'192.168.30.203';
Query OK, 0 rows affected (0.00 sec)
```

其中 select 对应的就是 priv_type 中的权限，mysql.user 对应 priv_level 中的 db_name.tbl_name，这是一个最简单的示例，当然啦，不使用 GRANT 语句，而通过 INSERT、UPDATE 方式修改字典表也是靠谱的！

话说 MySQL 数据库中有些权限的设计也很有意思，值得说道几句。

首先是关于 CREATE/DROP 这类权限，这是个很有意思的设定，拿 CREATE 权限来说，如果一个用户拥有了建库的权限，那么它也一定能创建表（但不能创建视图），此处的粒度设计没有那么细，MySQL 数据库并没有将建库和建表设计成两种权限，而是合二为一。DROP 权限也是类似的设计，不过与 CREATE 权限有所不同的是，DROP 权限也能删除视图对象。其实其他数据库中也有类似的设定，比如 Oracle 数据库环境，某个用户拥有创建对象的权限的话，那么它就一定拥有删除这个对象的权限。在 Oracle 看来，能创建就应该能删除，这两者是一体的，无法单独剥离。就我个人看来，Oracle 的设定明显更为老道，而且符合逻辑，MySQL 在这方面的设计还是显得规划有些不够清晰。

其次关于 ALL [PRIVILEGES]和 GRANT OPTION 权限，这是两种比较特殊的权限，甚至在授予或收回这两类权限时都不能与其他权限同时操作，并且这两个权限并不像它们名字显示的那样是“全部”的权限，后面的示例中会演示这一点。

最后关于 USAGE 权限，按照表 5-2 的描述中所示，这个权限的功能就是“没有权限”，但其实它不是完全没有权限，至少它还有一项权限，就是“登录权限”。对于使用 CREATE USER 语句创建的用户，该用户默认就会拥有 USAGE 权限，但是，又的确像描述中所说的那样，这个用户除了能够登录数据库，别的什么也做不了，因此就像是“没有权限”。

另外，还得再简要说一下 with_option 的几个选项：

- GRANT OPTION：允许用户再将该权限授予其他用户。
- MAX_QUERIES_PER_HOUR：允许用户每小时执行的查询语句数量。
- MAX_UPDATES_PER_HOUR：允许用户每小时执行的更新语句数量。
- MAX_CONNECTIONS_PER_HOUR：允许用户每小时连接的次数。
- MAX_USER_CONNECTIONS：允许用户同时连接服务器的数量。

这块的内容一看就是给用户设限制用的，我个人认为意义不大，不是说没有这类需求，而是这些选项的粒度仍然不够细致，不易碰到适合的应用场景。不过简单了解一下也是有必要的，万一哪天对某用户看着不爽，DBAer 心里应该明白，还是有法子限制该用户能够

使用的资源的。

其他部分就先不多说了，何况这个事儿也不能说得太细，主要是太细的东西三思也不懂，不懂装懂这个事儿俺脸皮虽然已经很厚，但做这类事儿的时候表情总是不够自然，不过请同学们放心，俺一定会继续努力，争取早日复制粘贴那谁的成功，用俺的真诚蒙到别人，蒙到所有的人……。

5.2.3　查看和收回用户权限

不管是授予还是收回用户的权限，通常首先需要知道用户当前都拥有什么权限。查询用户权限可以使用 SHOW GRANTS 语句，SHOW GRANTS 的语法比较简单，就一行：

```
SHOW GRANTS [FOR user]
```

其中 FOR user 还是个可选项，用于指定要查询的目标用户，如果不指定的话，则默认显示当前用户拥有的权限，效果等同于 SHOW GRANTS FOR CURRENT_USER()。

如果之前从未用过，那么 SHOW GRANTS 语句显示的结果可能会出乎意料，它返回的结果并不是某个权限类型的关键字，而是授权语句。

例如，查看用户 jss_grant@192.168.30.203 都拥有哪些权限，执行语句如下：

```
(system@localhost) [(none)]> show grants for jss_grant@192.168.30.203;
+-----------------------------------------------------------------------------------------------+
| Grants for jss_grant@192.168.30.203                                                           |
+-----------------------------------------------------------------------------------------------+
| GRANT USAGE ON *.* TO 'jss_grant'@'192.168.30.203' IDENTIFIED BY PASSWORD '*284578888014774
CC4EF4C5C292F694CEDBB5457' |
| GRANT SELECT ON 'jssdb'.* TO 'jss_grant'@'192.168.30.203'                                     |
| GRANT SELECT ON 'mysql'.'user' TO 'jss_grant'@'192.168.30.203'                                |
+-----------------------------------------------------------------------------------------------+
3 rows in set (0.00 sec)
```

从上述返回的结果可以看到，用户 jss_grant@192.168.30.203 拥有 3 项权限：查询 mysql.user 表、查询 jssdb 数据库下所有对象的权限以及登录 MySQL 数据库的权限。要我说，MySQL 数据库 SHOW GRANTS 语法最喜人之处在于，创建用户和授权语法都列出来了。

尽管前面我已经无数次提到过，直接查询 mysql 库中的数据字典表来修改或查看用户的权限信息，但我觉着如果是要查看某个用户的权限，使用 SHOW GRANTS 语句才是最好的方式，功能超强而且易用。

要收回用户权限，与之对应的命令是 REVOKE，它的语法从定义上分为两种：

```
REVOKE priv_type [(column_list)] [, priv_type [(column_list)]] ...
    ON [object_type] priv_level FROM user [, user] ...
REVOKE ALL PRIVILEGES, GRANT OPTION FROM user [, user] ...
```

前者用来处理指定的权限，有很多选项，这些选项的定义与 GRANT 中同名选项定义一模一样，这里不再赘述。后者功能较为独立，可以理解成专用于**清除**用户权限。

我们先来尝试收回 jss_grant@'192.168.30.203'用户，拥有的 mysql.user 表对象的 SELECT 权限，执行 REVOKE 命令如下：

```
(system@localhost) [(none)]> revoke select on mysql.user from jss_grant@192.168.30.203;
Query OK, 0 rows affected (0.00 sec)

(system@localhost) [(none)]> show grants for jss_grant@192.168.30.203;
+-----------------------------------------------------------------------------------------------+
| Grants for jss_grant@192.168.30.203                                                           |
+-----------------------------------------------------------------------------------------------+
| GRANT USAGE ON *.* TO 'jss_grant'@'192.168.30.203' IDENTIFIED BY PASSWORD '*284578888014774
CC4EF4C5C292F694CEDBB5457' |
| GRANT SELECT ON `jssdb`.* TO 'jss_grant'@'192.168.30.203'                                     |
+-----------------------------------------------------------------------------------------------+
2 rows in set (0.00 sec)
```

收回某个**普通**的指定权限立竿见影。注意，我说的是普通权限。有哪些权限不普通呢？如果此刻你的内心浮现出这个问题，说明看书不认真啊！在前面介绍 GRANT 语句时就曾提到过的，比如说 USAGE 权限，这个权限用户一经创建就会拥有，并且无法通过 REVOKE 语句收回。你要不相信哪，咱们来看一看：

```
(system@localhost) [(none)]> revoke usage on *.* from 'jss_grant'@'192.168.30.203';
Query OK, 0 rows affected (0.00 sec)
```

操作提示成功，但这是个假象，可以通过查看 jss_grant 用户拥有的权限来验证：

```
(system@localhost) [(none)]> show grants for jss_grant@192.168.30.203;
+-----------------------------------------------------------------------------------------------+
| Grants for jss_grant@192.168.30.203                                                           |
+-----------------------------------------------------------------------------------------------+
| GRANT USAGE ON *.* TO 'jss_grant'@'192.168.30.203' IDENTIFIED BY PASSWORD '*284578888014774
CC4EF4C5C292F694CEDBB5457' |
| GRANT SELECT ON `jssdb`.* TO 'jss_grant'@'192.168.30.203'                                     |
+-----------------------------------------------------------------------------------------------+
2 rows in set (0.00 sec)
```

毫无变化。前面 revoke 之所以没有报错，也跟 MySQL 在语句执行上的设定有关系。比方说，您可以尝试 revoke 任意权限 from user，它都不会报错的。之前曾提到过，MySQL 的返回就是这个德性：Query OK, 0 rows affected。

此外前面三思还提到过特殊的 ALL PRIVILEGES，这个权限也不像字面意义那么简单，所谓“所有权限”指的不是所有哟。你要不相信呀，咱们再来看一看，对 jss_grant@'192.168.30.203'用户执行 REVOKE ALL PRIVILEGES：

```
(system@localhost) [(none)]> revoke all privileges on *.* from jss_grant@'192.168.30.203';
Query OK, 0 rows affected (0.00 sec)
```

提示信息总是这样，我们能猜得到开头，不过能猜得对结局不？还是实际验证一下吧：

```
(system@localhost) [(none)]> show grants for jss_grant@192.168.30.203;
+-----------------------------------------------------------------------------------------------+
| Grants for jss_grant@192.168.30.203                                                           |
+-----------------------------------------------------------------------------------------------+
```


```
    | GRANT USAGE ON *.* TO 'jss_grant'@'192.168.10.203' IDENTIFIED BY PASSWORD
'*284578888014774CC4EF4C5C292F694CEDBB5457' |
    | GRANT SELECT ON `jssdb`.* TO 'jss_grant'@'192.168.30.203'                  |
    +-------------------------------------------------------------------------------------------+
    2 rows in set (0.00 sec)
```

又是毫无作用。那个特殊的 USAGE 权限就不说了，可是 ALL PRIVILEGES 居然连小小的普通的 SELECT 权限都没能收回，这还称得上 ALL 吗？呃，这个，称得上，它只是功能的设计并不像我们想象的那样而已。这几个知识点是 MySQL 数据库的权限体系设计上的细节，如不注意就有可能错误理解。

前后两个操作尽管看起来结果相同，但结论是完全不同的，前者是由于 USAGE 在 MySQL 权限体系中对于用户的特殊意义，后者是由于系统设计层的因素。MySQL 数据库中的权限，操作时授予和收回的权限级别（priv_level）必须对应，否则无法成功回收。

就上面这个例子中，授予 jss_grant@'192.168.30.203'用户 SELECT 权限时，是基于 jssdb 这样一个库级授予的，那么回收时，也必须明确指定是基于库级回收，如果指定 all on *.*，则无法收回 jssdb.*的权限，这也正是 MySQL 数据库权限粒度**分级**的特点。

因此，如果要让 REVOKE ALL PRIVILEGES 语句正确、有效地执行，就应该明确指定 on jssdb.*，例如：

```
    (system@localhost) [(none)]> revoke all privileges on jssdb.* from jss_grant@'192.168.30.203';
    Query OK, 0 rows affected (0.00 sec)

    (system@localhost) [(none)]> show grants for jss_grant@192.168.30.203;
    +-------------------------------------------------------------------------------------------+
    | Grants for jss_grant@192.168.30.203                                                       |
    +-------------------------------------------------------------------------------------------+
    | GRANT USAGE ON *.* TO 'jss_grant'@'192.168.30.203' IDENTIFIED BY PASSWORD
'*284578888014774CC4EF4C5C292F694CEDBB5457' |
    +-------------------------------------------------------------------------------------------+
    1 row in set (0.00 sec)
```

这下终于成功将权限收回了。

三思有过多年的 Oracle 数据库使用经验，在尝试使用和学习 MySQL 数据库期间感触很深，MySQL 数据库设计的确实很有特点，我想对于初学者，只要小脑袋瓜没有停止思考，一定会持续不断冒出各种各样的问题。对于用户的权限回收，经过前面一些演示，想必朋友们又会有新的疑问：若用户拥有各种不同级别、不同粒度、不同的权限，回收时难道也必须一一指定回收吗？这岂不是太过繁琐了。关于这一点，我可以负责任地说，把心踏踏实实搁肚子里头吧，MySQL 数据库就跟繁琐俩字不沾边，作为一款开源的轻量级数据库，MySQL 就没有什么复杂的特性，自然也不会有繁琐的操作。

对于前面这个疑问，如果确定要干净利索地清除某个用户的所有权限，并且还要保留这个用户（这是什么变态需求），那么，REVOKE 语句的第二种语法派上用场了：

```
    REVOKE ALL PRIVILEGES, GRANT OPTION FROM user
```

这是个固定语法，功能正是用于收回用户的所有权限，不管授予用户的是什么权限级

别什么对象的什么权限，一条语句执行下去，直接将用户恢复至裸身（USAGE）状态。

当前，jss_grant@'192.168.30.203'用户有各类权限如下：

```
(system@localhost) [(none)]> show grants for jss_grant@'192.168.30.203';
+-----------------------------------------------------------------------------------------------------------+
| Grants for jss_grant@192.168.30.203                                                                       |
+-----------------------------------------------------------------------------------------------------------+
| GRANT USAGE ON *.* TO 'jss_grant'@'192.168.30.203' IDENTIFIED BY PASSWORD '*284578888014774
CC4EF4C5C292F694CEDBB5457' |
| GRANT UPDATE, DELETE ON `jssdb`.* TO 'jss_grant'@'192.168.30.203'                                         |
| GRANT ALTER ON `jssdb_mc`.* TO 'jss_grant'@'192.168.30.203'                                               |
| GRANT SELECT ON `mysql`.`user` TO 'jss_grant'@'192.168.30.203'                                            |
+-----------------------------------------------------------------------------------------------------------+
4 rows in set (0.00 sec)
```

怎么一次性收回所有权限呢？执行 REVOKE 语句如下：

```
(system@localhost) [(none)]> revoke all,grant option from jss_grant@'192.168.30.203';
Query OK, 0 rows affected (0.00 sec)

(system@localhost) [(none)]> show grants for jss_grant@192.168.30.203;
+-----------------------------------------------------------------------------------------------------------+
| Grants for jss_grant@192.168.30.203                                                                       |
+-----------------------------------------------------------------------------------------------------------+
| GRANT USAGE ON *.* TO 'jss_grant'@'192.168.30.203' IDENTIFIED BY PASSWORD '*284578888014774
CC4EF4C5C292F694CEDBB5457' |
+-----------------------------------------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

你看，你看，用户的权限悄悄地在改变。

5.2.4　删除用户

我想不出一个仅拥有 USAGE 权限的用户存在的意义，干脆，删了它吧！怎么，您担心会丢失数据、影响系统稳定，放心吧！一个失势的人对组织是没什么危险的，只要权利被收回，它就立刻什么都不是。

很多人之所以担心删用户会丢数据，主要是受其他数据库产品的影响，比如说 Oracle 中删除用户（或其他对象，比如表空间），如果该用户下有很多的对象，那么删除用户的同时也会把这些对象及关联的数据统统删除，尽管 Oracle 会人性化地提醒你删除的用户下仍然存在数据，但如果强制级联删除（附加 CASCADE 选项），那么该删就还是删了。

MySQL 的删除用户语法中就不存在 CASCADE 的选项，为什么不存在呢？并不是 MySQL 对数据的保护不如 Oracle 那么上心，而是由最重要的一条与 Oracle 不同的机制决定。MySQL 数据库中的对象保存并不是依赖于用户，而是依赖于库（database），用户被删除没有任何关系，对象仍在，好好地保存在存储它的数据库中。

因此，MySQL 数据库中的用户删了就删了，如果外部应用不使用该用户的话，那么

我们可以认为该用户被删除无影响。即使发现真的删错了，想给它恢复身份的话也很简单，这不就是一句话的事儿嘛，只要重新向 mysql.user 表插入记录（注册建档），并授予所需权限即可（授予官阶），至于底层数据的意见那是完全可以忽视的。

看我说的这么笃定，下面就实际删个试试吧！MySQL 中删除用户的语法非常简单：

```
DROP USER user [, user] ...
```

从语法上大家想必也都看出来了，可以一次性删除多个用户，这里三思就准备一步删除之前创建的 jss_grant、jss_insert 和 jss_ip 几个用户，执行 DROP USER 命令如下：

```
(system@localhost) [(none)]> drop user jss_grant@192.168.30.203, jss_insert@192.168.30.203,
jss_ip@192.168.30.203;
Query OK, 0 rows affected (0.00 sec)
```

需要说明的一点是，DROP USER 不会自动中止已连接的用户会话，也就是说被删的用户如果在删前已经连接上了服务器，并且连接尚未中断，那它此时还能继续执行一定的操作，只是它的身份已经变成了黑户。

5.3　权限级别

总的来说，MySQL 数据库的权限从大的粒度上可以分成 5 类：全局、数据库、表、列、程序。通过对这 5 个大类权限的细分，可以精确地为**某个**用户分配从**某台**机器连接进来访问**某个**数据库下**某个**表的**某个**列的**某部分**记录权限。

授权主要是通过 GRANT 命令（或手动向字典表中插入或修改记录），对应的权限关键字，就是 5.2 节中所列的 priv_type，相对于 Oracle 数据库来说，我个人认为，MySQL 数据库中权限设定真简单。注意，简单不是一个贬义词，三思曾经无数次在无数个场合强调过这样一种观点：简单意味着灵活，而灵活在有心人的手上能实现的功能非常之强大。

本章尽可能多的通过示例，帮助大家理解 GRANT 语句的用法，当然，最重要的是理解 MySQL 数据库的权限体系，考虑到 MySQL 中的各级权限主要基于若干个字典表，因此本段介绍时会将这几个字典表的结构列为重要参照。

提　示

user/db/host 几个字典表中，host 列的值对大小写不敏感。User、Password、Db 和 Table_name 几个列值对大小写敏感。Column_name 列值对大小写不敏感。

5.3.1　全局

全局这个词儿一听就知道层次很高，宏观的事物都很重要，你看播音员每当提到宏观（经济数据）都是一脸的肃穆，连那个号称 60 年没出过一条假新闻的著名报纸，发表宏观经济数据时都兴奋得跟打了鸡血似的，不是喊保 8 就是喊超 9，虽然我怎么也闹不明白 8

和 9 到底是什么情况。

具体到 MySQL 这样一款小软件，全局这个级别也差不到哪儿去，我就说一条，与全局相关的权限信息记在 mysql.user 表中。这下大家知道厉害了吧，mysql.user 表对象里是等闲数据能待的地方吗？前面提到过，这个表管**登录**，控制用户能不能连接这样一等一的最重要的事情，闲杂记录能保存在这里吗？但是我们也要注意了，这个全局权限可不一定就能拥有所有的权限，它具体指的是能够拥有该 MySQL 服务器**所有**数据库的[**所有**]对象的[**所有**]权限（注：[]表示可选）。

下面新创建一个用户，并授予它 CREATE 权限，代码如下：

```
(system@localhost) [(none)]> grant create on *.* to jss_global;
Query OK, 0 rows affected (0.00 sec)

system@localhost) [(none)]> select * from mysql.user where user='jss_global'\G
*************************** 1. row ***************************
                  Host: %
                  User: jss_global
              Password:
           Select_priv: N
           Insert_priv: N
           Update_priv: N
           Delete_priv: N
           Create_priv: Y
             Drop_priv: N
           Reload_priv: N
.........
.........
.........
1 row in set (0.00 sec)
```

查看返回的 mysql.user 表中记录的信息，所有与权限相关列的列值多为'N'，表示没有权限，只有被授予了全局操作权限，mysql.user 表中权限对应列值才是'Y'，这种情况下，该用户就拥有在所连接的 MySQL 服务器下所有数据库中执行相应操作的权限。

以前面创建的 jss_global 用户为例，授予了 CREATE 权限之后，该用户即可轻松**查看**（没错，不仅能创建，还能查看）当前连接的 MySQL 数据库中创建的所有数据库，并能够在任意数据库中创建表对象（information_schema 库除外，该库具有一定特殊性，后面章节详述）。

找台客户端，以 jss_global 用户登录，由于创建时没有指定主机和密码信息，因此可以从任意主机并且无需输入任何密码登录：

```
[mysql@mysqldb02 ~]$ mysql -ujss_global -h 192.168.30.243
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 17
Server version: 5.6.12-log JSS for mysqltest
.........
.........
(jss_global@192.168.30.243) [(none)]>
```

执行 SHOW DATABASES 命令，可以查看当前存在的所有数据库：

```
(jss_global@192.168.30.243) [(none)]> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| jssdb              |
| jssdb_mc           |
| mysql              |
| performance_schema |
+--------------------+
5 rows in set (0.00 sec)
```

创建表对象，也没有问题，如在 jssdb 库中创建一个名为 test1 的表对象：

```
(jss_global@192.168.30.243) [(none)]> use jssdb;
Database changed
(jss_global@192.168.30.243) [(jssdb)]> create table test1 (v1 varchar(20));
Query OK, 0 rows affected (0.00 sec)
```

创建成功，大家可以通过 SHOW TABLES 命令或 DESC 命令查看验证，这里不演示了。

MySQL 数据库中权限的设计自有其逻辑，有些设定符合人们（谨代表我个人）的常规思维，有些则跟我们下意识的认知有较大差距。

就拿刚刚演示的这个 CREATE 权限来说，当用户拥有全局的 CREATE 权限，那么它同时也级联拥有了查看所有数据库（SHOW DATABASES）和数据库下所有对象的权限，能建就能看，这点容易理解；但是，明确授予的 CREATE 权限，又确实只拥有“创建”的权限，想删除对象是不行的，哪怕这个对象就是它刚刚创建的：

```
(jss_global@192.168.30.243) [jssdb]> drop table test1;
ERROR 1142 (42000): DROP command denied to user 'jss_global'@'192.168.30.203' for table 'test1'
```

修改也不行：

```
(jss_global@192.168.30.243) [jssdb]> alter table test1 add (v1 varchar(20));
ERROR 1142 (42000): ALTER command denied to user 'jss_global'@'192.168.30.203' for table 'test1'
```

甚至连查询都不行：

```
(jss_global@192.168.30.243) [jssdb]> select * from test1;
ERROR 1142 (42000): SELECT command denied to user 'jss_global'@'192.168.30.203' for table 'test1'
```

现在您明白了吧，他授予的仅仅只是 CREATE 权限，想删除是不行的，连查看都是肯定不行的。对对，即使要删除的对象是自己刚刚创建的也不行。不不，多贵的计算机都不行。

全局这么重要的粒度，能够在这一级授予的权限自然不少，在 5.2.2 小节中提到的权限大部分都可以在全局级授予，与 MySQL 服务管理相关的权限则全部是在全局级进行设置。表 5-3 罗列了可在全局粒度授予的权限，以及该权限与 mysql.user 字典表列的对应关系。

表 5-3　全局权限列表

user 字典表列名	对应权限名
select_priv	SELECT
insert_priv	INSERT
update_priv	UPDATE
delete_priv	DELETE
create_priv	CREATE
drop_priv	DROP
reload_priv	RELOAD
shutdown_priv	SHUTDOWN
process_priv	PROCESS
file_priv	FILE
grant_priv	GRANT OPTION
references_priv	REFERENCES
index_priv	INDEX
alter_priv	ALTER
show_db_priv	SHOW DATABASES
super_priv	SUPER
create_tmp_table_priv	CREATE TEMPORARY TABLES
lock_tables_priv	LOCK TABLES
execute_priv	EXECUTE
repl_slave_priv	REPLICATION SLAVE
repl_client_priv	REPLICATION CLIENT
create_view_priv	CREATE VIEW
show_view_priv	SHOW VIEW
create_routine_priv	CREATE ROUTINE
alter_routine_priv	ALTER ROUTINE
create_user_priv	CREATE USER
event_priv	EVENT
trigger_priv	TRIGGER
create_tablespace_priv	CREATE TABLESPACE

提 示

默认情况下，使用 CREATE USER 创建的用户，能够登录 MySQL 数据库，并且还具有操作 test 库中对象的权限，这是 MySQL 数据库的默认设定，关于 test 数据库的权限问题，将在后面章节中专门描述。

5.3.2　数据库

数据库级别的权限，主要用于控制账户（'user'@'host'）操作某个数据库的权限，在这一粒度对用户做了授权后，用户就拥有了该数据库下[所有]对象的[所有]权限。

数据库级别的权限信息记录在 mysql.db 表。在介绍 mysql.db 表之前，三思想先特别提一下 mysql.host 表，这个表也与数据库粒度的权限有关联，它的功能相对奇特，是用于控制某些主机（host）是否拥有操作某个数据库的权限，在可设置的权限方面跟 mysql.db 几乎一模一样。

mysql.host 表在 MySQL 5.5 及之前版本中的处境很特别，默认情况下 GRANT/REVOKE 语句并不触发对该表数据的读、写，因此多数情况下该表都没啥用，极易被忽略。不过在应对某些特定场景下，DBA 可以手动操作（insert、update、delete）该表来实现某些特殊的需求。比如说只希望某些主机拥有操作某个数据库的权限时，mysql.user 完全派不上用场（它是针对全局的嘛，管不到 db 这么细的粒度），那么使用 mysql.host 就可以轻松实现，因为该表对权限的验证正是使用 host 这个维度。

当然啦，这个需求使用 mysql.db 表也可以实现，mysql.db 表是通过 user+host 两个维度来验证权限，比 mysql.host 多了一个维度，不过由于 MySQL 数据库的权限字典表能够支持通配符，并且 user 列可以为空（代表所有用户），通过灵活设置也可以实现 mysql.host 表的功能。我想也正是基于此，从 5.6 版本开始，mysql.host 表已被明确废弃。不过如果您在使用之前版本的数据库，恰好场景适当，倒是仍可以用用 mysql.host 表。

还是回到 mysql.db 表吧，功能前头已经说过了，不过出于加深印象的目的，我再重复说一遍大家没什么意见吧，有意见也不要紧，我的邮箱地址网上都写着哪，有啥抱怨的话尽管发，Gmail 邮箱，空间有好个 G 哪。

我个人感觉将数据库级权限与全局级权限对比起来更好理解，全局级权限大家都知道了吧，用来控制用户操作所有数据库的权限（以及管理 MySQL 服务的权限），数据都是保存在 mysql.user 字典表中。若只希望授予用户操作某个数据库的权限，该怎么办呢？那就该 mysql.db 出马啦！你要问 mysql.user 和 mysql.db 差在哪儿，对比一下两个字典表的表结构您就明白啦（表 5-4）。

表 5-4　全局和库级权限对应表

mysql.user 表	mysql.db 表
Host	Host
User	User
Password	
	Db
Select_priv	Select_priv
Insert_priv	Insert_priv
Update_priv	Update_priv
Delete_priv	Delete_priv
Create_priv	Create_priv
Drop_priv	Drop_priv
Reload_priv	Grant_priv
References_priv	References_priv
Index_priv	Index_priv
Alter_priv	Alter_priv
Create_tmp_table_priv	Create_tmp_table_priv
Lock_tables_priv	Lock_tables_priv
Create_view_priv	Create_view_priv
Show_view_priv	Show_view_priv
Create_routine_priv	Create_routine_priv
Alter_routine_priv	Alter_routine_priv
Execute_priv	Execute_priv
Event_priv	Event_priv
Trigger_priv	Trigger_priv
Shutdown_priv	
Process_priv	
.....	
.....	
.....	

你看，mysql.db 表中有的列，在 mysql.user 中几乎全都有，而 mysql.user 中有的列则有一堆 mysql.db 表中都不存在呀。看看前面章节中介绍的权限说明，多出的列正是 MySQL 服务级的管理权限，说 mysql.db 是 mysql.user 表的子集都不为过。mysql.db 与 mysql.user 相比多出的 Db 列，不正是用来指定要管理的目标数据库嘛！

授予用户某个数据库的管理权限，执行 GRANT 语句时，相比全局就得缩小授权范围，把全局时指定的*.*改成 dbname.*就行啦！例如，创建 jss_database 用户，并授予 jssdb 库下创建对象的权限，执行命令如下：

```
(system@localhost) [(none)]> grant create on jssdb.* to jss_db;
Query OK, 0 rows affected (0.00 sec)
```

创建成功，查看 jss_db 用户在各字典表的记录明细，以便我们能够更清晰地理解权限字典表在用户权限环境所起到的作用。

先来看看刚刚创建的用户，在 mysql.user 全局权限表中的信息：

```
(system@localhost) [(none)]> select * from mysql.user where user='jss_database'\G
*************************** 1. row ***************************
                 Host: %
                 User: jss_database
             Password:
          Select_priv: N
          Insert_priv: N
          Update_priv: N
          Delete_priv: N
          Create_priv: N
            Drop_priv: N
          Reload_priv: N
        Shutdown_priv: N
.............
.............
```

操作类权限都是 N（相当于仅拥有 USAGE 权限），这就对了，允许该用户登录 MySQL 数据库。那么操作 jssdb 数据库的权限写在哪了呢？再看看 mysql.db 库级权限字典表吧：

```
(system@localhost) [(none)]> select * from mysql.db where user='jss_db'\G
*************************** 1. row ***************************
                 Host: %
                   Db: jssdb
                 User: jss_db
          Select_priv: N
          Insert_priv: N
          Update_priv: N
          Delete_priv: N
          Create_priv: Y
            Drop_priv: N
           Grant_priv: N
      References_priv: N
           Index_priv: N
           Alter_priv: N
Create_tmp_table_priv: N
     Lock_tables_priv: N
     Create_view_priv: N
       Show_view_priv: N
  Create_routine_priv: N
   Alter_routine_priv: N
```


```
            Execute_priv: N
              Event_priv: N
            Trigger_priv: N
1 row in set (0.00 sec)
```

这下就比较清晰了，Db 表显示了可操作的库名，Create_priv 列值显示 Y，表示这个用户拥有指定库中对象的创建权限。

下面再通过该用户连接到 MySQL 数据库中看一下吧！使用 jss_db 用户登录，查看当前可访问的数据库：

```
(jss_db@192.168.30.243) [(none)]> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| jssdb              |
+--------------------+
2 rows in set (0.00 sec)
```

jssdb 库倒是列出来了，但是，好奇怪呀！不是说只授予了 jssdb 库的权限吗，怎么还能看到 information_schema 库呢？别急，咱们马上就会提到这一点。

1．*并不存在的* INFORMATION_SCHEMA *库*

熟悉 Oracle 数据库的朋友想必知道，在 Oracle 数据库中有一堆 v$*视图、user_*、all_*等字典表，所有能够成功连接数据库的用户都可以访问这些对象（无需额外授权）。MySQL 数据库中也存在一系列这样的对象，比如 TABLES、VIEWS、COLUMNS 等等，所有能够成功登录到 MySQL 数据库的用户都能访问。

想一想，这些对象在哪呢？没错，正是在 INFORMATION_SCHEMA 数据库下。既然这类对象能够被访问，那么 INFORMATION_SCHEMA 库自然也就能被所有用户看到啦，这样才符合逻辑。

需要注意的是，MySQL 中的 INFORMATION_SCHEMA 并不是真正的数据库，在操作系统层并没有与之对应的物理文件，这个数据库及库中的对象全是由 MySQL 自动维护的一系列虚拟对象，这些对象用户能看却不能改（不能直接改），并且与 Oracle 数据库中的数据字典表类似，用户查询这些对象中的记录时，看到的都是自己有权限看到的对象。比如说拥有 jssdb 库创建权限的 jss_db 用户，能够在 INFORMATION_SCHEMA 数据库的 TABLES/COLUMNS 等对象中，查看 jssdb 库中所有表和列的信息，但是因为没有视图、过程这类对象的操作权限，访问 VIEWS 字典表时，就查看不到记录啦！

INFORMATION_SCHEMA 库中对象的另一特殊之处在于，用户不能对 INFORMATION_SCHEMA 数据库中的对象做授权。比如将 information_schema.tables 表对象的 SELECT 权限授予某个用户，这样操作肯定会失败，即使是管理员用户也不行。

2．*有趣的* test *库*

除了 INFORMATION_SCHEMA 这样的虚拟库外，MySQL 数据库中的 test 库的默认权

限也需要引起 DBA 们注意。

新建 MySQL 数据库后，默认创建的 test 数据库权限比较怪异，所有可连接的用户都能够拥有权限访问该库，并操作其中的对象。这是怎么实现的呢？其实很简单，查看库级权限字典表 mysql.db，您就明白了：

```
mysql> select * from mysql.db where db like 'test%'\G;
*************************** 1. row ***************************
                 Host: %
                   Db: test
                 User:
          Select_priv: Y
          Insert_priv: Y
          Update_priv: Y
          Delete_priv: Y
          Create_priv: Y
            Drop_priv: Y
           Grant_priv: N
      References_priv: Y
           Index_priv: Y
           Alter_priv: Y
Create_tmp_table_priv: Y
     Lock_tables_priv: Y
     Create_view_priv: Y
       Show_view_priv: Y
  Create_routine_priv: Y
   Alter_routine_priv: N
         Execute_priv: N
           Event_priv: Y
         Trigger_priv: Y
*************************** 2. row ***************************
                 Host: %
                   Db: test\_%
                 User:
          Select_priv: Y
          Insert_priv: Y
          Update_priv: Y
          Delete_priv: Y
          Create_priv: Y
            Drop_priv: Y
           Grant_priv: N
      References_priv: Y
           Index_priv: Y
           Alter_priv: Y
Create_tmp_table_priv: Y
     Lock_tables_priv: Y
     Create_view_priv: Y
       Show_view_priv: Y
  Create_routine_priv: Y
   Alter_routine_priv: N
```


```
            Execute_priv: N
              Event_priv: Y
            Trigger_priv: Y
2 rows in set (0.00 sec)
```

你看，从权限上来看，Host 为%，User 为空，这就说明了不限制的，所有能连接到 MySQL 的用户，全都拥有 test 及 test 开头的数据库的几乎所有权限。

这无疑存在安全上的隐患，先不说在其中创建的重要对象可被任何人访问，就算该库中没有任何对象，假如有人想恶意破坏 DB 服务，只要登录数据库后，在该库创建一个超大对象，把空闲空间全部占满，就相当于变相达到了破坏 DB 服务的目的。对于这类权限没啥好客气的，该咋处理就咋处理吧！

不过如果读者朋友是按照三思在本书中介绍的步骤创建数据库，那就不会存在这种隐患了，还记得第 3 章中配置数据库环境时我们做过的操作吗：

```
(root@localhost) [(none)]> truncate table mysql.db;
Query OK, 0 rows affected (0.00 sec)
```

直接清空 mysql.db 表中记录，这两个权限已被删除，隐患早已经被排除啦！

顺便提出一个问题，如果想让所有用户都拥有访问 jssdb 库中对象的权限，GRANT 语句应该怎么写呢？有兴趣的朋友不妨在自己的测试环境中模拟一下吧！

5.3.3　表

表作为具体的对象，当我们谈论到对某个对象授权时，已经进入到一个相对细粒度的权限级别了，表对象的授权信息保存在 mysql.tables_priv 字典表中。

我知道很多初学者在学习 MySQL 权限操作时，由于对权限体系了解有限不够熟悉，甚至可能不清楚究竟有哪些权限可供授权。这个问题解决起来也很简单，直接看官方文档，文档中的表 5-2 罗列了所有可授予的权限。当然啦看本书也靠谱，前面 5.2.2 节中列的表就是抄（提起这个字儿我脸就红了）自官方文档的权限列表，里面各种权限明细写得清清楚楚明明白白，看完后记住就不会再迷茫啦！

还有些朋友文档也没少看，可就是记不住，一方面由于权限类型多（其实 MySQL 中的权限相比 Oracle 已经少太多了），另一方面权限还分了多个粒度，谁能记得清哪个粒度都有哪些权限哪。针对这种情况也很好解决，用 desc 查看相关表对象的结构即可。

比方说，现在咱们都不知道在表一级，究竟能够授予用户什么样的权限（或者说用户有什么样的选择），那么直接使用 desc mysql.tables_priv 查看，例如：

```
(system@localhost) [(none)]> desc mysql.tables_priv;
+-------------+--------------+------+-----+----------+------------+
| Field       | Type         | Null | Key | Default  | Extra      |
+-------------+--------------+------+-----+----------+------------+
| Host        | char(60)     | NO   | PRI |          |            |
| Db          | char(64)     | NO   | PRI |          |            |
| User        | char(16)     | NO   | PRI |          |            |
| Table_name  | char(64)     | NO   | PRI |          |            |
```


```
| Grantor     | char(77)      | NO   | MUL |           |             |
| Timestamp   | timestamp     | NO   |     | CURRENT_TIMESTAMP | on update CURRENT_TIMESTAMP |
| Table_priv  | set('Select','Insert','Update','Delete','Create','Drop','Grant','References',
  'Index','Alter','Create View','Show view','Trigger') | NO   |      |           |         |
| Column_priv | set('Select','Insert','Update','References')          | NO   |     |    |    |
+-------------+---------------+------+-----+---------+-----------+
8 rows in set (0.00 sec)
```

直接输出的信息较长，这里做了些删减。简要说一下 tables_priv 字典表的结构：

- Host：来源主机。
- Db：对象所属数据库。
- User：用户名。
- Table_name：表对象名称。
- Grantor：执行权限授予的用户。
- Timestamp：授予权限的时间。
- Table_priv：能够授予的表粒度的权限，也就是我们最关注的信息。
- Column_priv：能够授予的列粒度的权限。

Host+Db+User+Table_name 四个维度的共同作用成就一条权限，粒度够细。

注意看 Table_priv、Column_priv 两列对应的列值，这些列值就是表对象能够授予的权限。这下知道权限关键字怎么写了吧？三思老早就表达过这样一种观点，学习是有技巧的，死记硬背（SJYB）是技巧之一，但不一定是最好的，随机应变（SJYB）才是……。

知道了关键字，就可以根据需求进行授权了。比如说，向 jss_tables 用户授予 users 表的全部权限，该怎么写 GRANT 语句呢：

```
(system@localhost) [(none)]> grant all on jssdb.users to jss_tables;
Query OK, 0 rows affected (0.02 sec)
```

哎哟哟，咋没写前面 desc 里看到的权限关键字呢？这样写也能成功授权吗？嘿嘿，三思都说了要 SJYB 的嘛，让事实来说话吧：

```
(system@localhost) [(none)]> select * from mysql.tables_priv where user='jss_tables' and table_
name='users'\G
*************************** 1. row ***************************
       Host: %
         Db: jssdb
       User: jss_tables
 Table_name: users
    Grantor: system@localhost
  Timestamp: 0000-00-00 00:00:00
 Table_priv:      Select,Insert,Update,Delete,Create,Drop,References,Index,Alter,Create      View,Show
view,Trigger
Column_priv:
1 row in set (0.00 sec)
```

这下理解了吧，ALL 就是所有权限嘛！

不过细心的朋友想必早已注意到 tables_priv 表中的 Columns_priv 列了吧？你说表粒度

的权限怎么会出现在对列级权限的指定中呢？这点在我看来其实正是体现 MySQL 细节设计上的特点，作为一款开源软件，它在整体设计上有时确实让人感觉摸不着头脑，说的更直白些就是它自己都没想清楚啊！这一点不仅仅体现在权限设计上，在其他设计比如初始化参数、管理功能等都有体现。

总之就是 Column_priv 列在声明表级权限时没用，但在授予列级权限时就有反应了，继续往下看吧！

5.3.4　列

列级权限是 MySQL 权限体系中的最细粒度，属于权限认证体系中的高精尖武器。通过对表中列的授权，可以实现只允许从某主机来的某用户访问某库的某表的某列。

列级权限保存在 mysql.columns_priv 字典中，该字典结构如下：

```
(system@localhost) [(none)]> desc mysql.columns_priv;
+-------------+---------------+------+-----+-----------+-----------+
| Field       | Type          | Null | Key | Default   | Extra     |
+-------------+---------------+------+-----+-----------+-----------+
| Host        | char(60)      | NO   | PRI |           |           |
| Db          | char(64)      | NO   | PRI |           |           |
| User        | char(16)      | NO   | PRI |           |           |
| Table_name  | char(64)      | NO   | PRI |           |           |
| Column_name | char(64)      | NO   | PRI |           |           |
| Timestamp   | timestamp     | NO   |     | CURRENT_TIMESTAMP | on update CURRENT_TIMESTAMP |
| Column_priv | set('Select','Insert','Update','References') | NO   |     |           |           |
+-------------+---------------+------+-----+-----------+-----------+
7 rows in set (0.00 sec)
```

列级权限需要 Host+Db+User+Table_name+Column_name 五个粒度，另外从上面的对象结构可以看出，对于列级权限可授予的共有 4 项，其中只有前 3 项有实际意义：

- Select：查询权限。
- Insert：插入权限。
- Update：修改权限。
- References：尚未应用，直接无视。

授予列级权限，在执行 GRANT 语句时，语法上稍有不同，主要体现在指定列级的粒度语法并不在 ON 子句，而是在指定 priv_type 时顺道附带列名的方式。例如，授予 jss_cols 用户查询 jssdb.users 表 phoneno 列的权限，执行语句如下：

```
(system@localhost) [(none)]> grant select (phoneno) on jssdb.users to jss_cols;
Query OK, 0 rows affected (0.00 sec)
```

查看字典表中存储的信息：

```
(system@localhost) [(none)]> select * from mysql.columns_priv;
+------+-------+----------+------------+-------------+---------------------+-------------+
| Host | Db    | User     | Table_name | Column_name | Timestamp           | Column_priv |
+------+-------+----------+------------+-------------+---------------------+-------------+
```


```
| %    | jssdb | jss_cols | users      | phoneno     | 0000-00-00 00:00:00 | Select      |
+------+-------+----------+------------+-------------+---------------------+-------------+
1 row in set (0.00 sec)

(system@localhost) [(none)]> select * from mysql.tables_priv where user='jss_cols';
+------+-------+----------+------------+-------------+---------------------+------------+-------------+
| Host | Db    | User     | Table_name | Grantor     | Timestamp           | Table_priv | Column_priv |
+------+-------+----------+------------+-------------+---------------------+------------+-------------+
| %    | jssdb | jss_cols | users      | system@localhost | 0000-00-00 00:00:00 |    | Select  |
+------+-------+----------+------------+-------------+---------------------+------------+-------------+
1 row in set (0.00 sec)
```

列级字典表中是有数据的，表级字典表中也是有记录存在的，就目前的实际情况来看，column_priv 表控制具体的权限，table_priv 表中的数据则是用来标记该条授权的一些基础信息，比如授予者、操作时间等。

对同一个表对象再授权另一个权限，看看字典表中如何存储就更加明确了：

```
(system@localhost) [(none)]> grant insert (address) on jssdb.users to jss_cols;
Query OK, 0 rows affected (0.00 sec)

(system@localhost) [(none)]> select * from mysql.columns_priv;
+------+-------+----------+------------+-------------+---------------------+-------------+
| Host | Db    | User     | Table_name | Column_name | Timestamp           | Column_priv |
+------+-------+----------+------------+-------------+---------------------+-------------+
| %    | jssdb | jss_cols | users      | phoneno     | 0000-00-00 00:00:00 | Select      |
| %    | jssdb | jss_cols | users      | address     | 0000-00-00 00:00:00 | Insert      |
+------+-------+----------+------------+-------------+---------------------+-------------+
2 rows in set (0.00 sec)

(system@localhost) [(none)]> select * from mysql.tables_priv where user='jss_cols';
+------+-------+----------+------------+-------------+---------------------+------------+----------------+
| Host | Db    | User     | Table_name | Grantor     | Timestamp           | Table_priv | Column_priv    |
+------+-------+----------+------------+-------------+---------------------+------------+----------------+
| %    | jssdb | jss_cols | users      | system@localhost | 0000-00-00 00:00:00 |    | Select,Insert |
+------+-------+----------+------------+-------------+---------------------+------------+----------------+
1 row in set (0.00 sec)
```

注意到差别了吧！tables_priv 只是表级粗粒度的记录，columns_priv 才是决定列级权限粒度的核心。

下面使用刚刚创建的 jss_cols 用户连接到数据库查看一下：

```
[mysql@mysqldb02 ~]$ mysql -ujss_cols -h 192.168.30.243
Welcome to the MySQL monitor.  Commands end with ; or \g.
.....
.....
(jss_cols@192.168.30.243) [(none)]> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
```


```
| jssdb              |
+--------------------+
2 rows in set (0.00 sec)

(jss_cols@192.168.30.243) [(none)]> use jssdb;
Database changed
(jss_cols@192.168.30.243) [jssdb]> show tables;
+-----------------+
| Tables_in_jssdb |
+-----------------+
| users           |
+-----------------+
1 row in set (0.00 sec)

(jss_cols@192.168.30.243) [jssdb]> desc users;
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| address | varchar(50) | YES  |     | NULL    |       |
| phoneno | varchar(15) | YES  |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+
2 rows in set (0.02 sec)
```

能看，且仅能查看授权了的表的指定列，看起来该表似乎只有这两个列，其实是因为它只能看到这两列，实际操作时也将发现，这两列的权限都是不一样的。

问：怎么查看当前用户拥有的权限呢？

答：朋友，可还记得 5.2.3 节讲过的 SHOW GRANTS 命令。

这里需要注意的一点是，尽管通过查看表结构，或者是使用 SELECT 语句查询表数据时只能查到被授予权限的列，但是，该用户查询 information_schema.tables 或其他相关字典表时，看到的表的信息仍然是完整的，比如表的大小、索引大小、平均列长度等，这也是 information_schema 库比较特殊的另一个体现吧！

5.3.5　程序

MySQL 中的程序（ROUTINE）主要是指 procedure 和 function 两类对象，这两类对象的权限与前面描述的 4 种基本无关联（如果说有的话，也只是用户是否拥有连接数据库的权限），相对比较独立。

对于已存在的 Procedure/Function，DBA 可以对用户授予执行（EXECUTE）、修改（ALTER ROUTINE）、授予（GRANT）权限，这部分权限体现在 mysql.procs_priv 表中，例如：

```
(system@localhost) [(none)]> desc mysql.procs_priv;
+--------------+---------------------------------------+------+-----+-----------+--------+
| Field        | Type                                  | Null | Key | Default   | Extra  |
+--------------+---------------------------------------+------+-----+-----------+--------+
| Host         | char(60)                              | NO   | PRI |           |        |
| Db           | char(64)                              | NO   | PRI |           |        |
```


```
| User         | char(16)                                | NO   | PRI |           |         |
| Routine_name | char(64)                                | NO   | PRI |           |         |
| Routine_type | enum('FUNCTION','PROCEDURE')            | NO   | PRI | NULL      |         |
| Grantor      | char(77)                                | NO   | MUL |           |         |
| Proc_priv    | set('Execute','Alter Routine','Grant') | NO   |     |           |         |
| Timestamp    | timestamp          | NO   |     | CURRENT_TIMESTAMP | on update CURRENT_TIMESTAMP |
+--------------+-----------------------------------------+------+-----+----------+--------+
8 rows in set (0.00 sec)
```

此外，还可以授予用户创建（CREATE ROUTINE）权限，这个权限在 user、db、host 几个表中都有体现。拥有 CREATE ROUTINE 权限的用户能够创建 procedure、function 对象。这个权限是用户/库一级权限，而 EXECUTE、ALTER ROUTINE、GRANT 这 3 个权限则是对象级，都是针对某个指定的 procedure、function 做授权。

关于“程序”对象的权限操作就不演示了，实在是跟之前的权限授予、收回操作没啥区别，重复的事情做起来实在没意思，还浪费纸张，很不低碳，咱们还是接着做点对全人类有益的事情，少说点儿废话，多做点儿实事儿吧！

5.4　账户安全管理

提到账户安全，我想即使是从没有接触过 MySQL 软件的朋友，只要使用过计算机，畅游过互联网就都会有自己的理解，甚至对于那些从没接触过计算机的人也有自己的心得。毕竟当下就是信息时代，不管是作为普通人到银行办理业务，还是网民在网上购物消费，都少不得要管理一系列的账户和口令，因此可以说人人都对此早有接触，并且积累下深厚的账户管理经验。

因为关于账户安全的一般性原则是通用的，因此这里三思并不想再去强调什么密码设置原则、口令保护指南等陈词滥调。本章着重谈到的几方面内容更多可以视为一些小技巧，希望能够真正为大家在运维 MySQL 数据库的过程中，提供一些参考，减少一些隐患。

5.4.1　用户与权限设定原则

看完前面的内容，我想大家已经对如何创建用户、如何管理用户的权限有了认识。不过，要知道想炒出一盘可口的佳肴，只有原料可还不行，厨艺的高低更加重要。

创建出一个数据库账户，权限是给大还是给小；前端的应用服务很多，是共用同一个账户，还是分开独立操作；每一种选择都有它的优点，相应也会有它的弊端。究竟怎么设定才对，有时候我觉着这就像问大厨炒菜时放多少盐合适，它的回答可能是：适量就好。这个答案听起来很虚，但实际上有时候就是这么微妙，它确实不是一个定量，需要大家根据实际情况调整。

本节的内容并不一定都对，更多是三思个人的理解，不过，我可以负责任地告诉大家，

我所管理的数据库的账户，也正是按照这些原则在维护。

总体规划目标：

- 考虑实际情况及使用习惯，权限的设定应该是在尽可能不增加应用端开发工作量的前提下，尽可能缩小权限分配的粒度。
- 做到业务级的账户分离，不同应用（项目）分别使用不同的账户执行操作。
- 使操作者执行 SQL 时所使用的账户，与其能够执行的操作相对应。
- 降低误操作的几率，保障数据库系统安全。

权限按照可控程度分级设定。如何定义级别要根据实际业务规模，我们目前的环境综合考虑了团队规模和业务数据规模，最终确定以库为单位创建账户，在达到安全设定目标的前提下，尽可能简化流程，将权限级别设为 3 级：

- {user}_oper：定义为操作用户，拥有指定库下所有对象的操作权限，授予增加（INSERT）、删除（DELETE）、修改（UPDATE）、查询（SELECT）记录的权限，主要用于前端应用程序，连接数据库读写数据。
- {user}_read：定义为只读用户，拥有指定库下对象的读取权限，授予查询（SELECT）记录的权限，可用于数据查询、SQL 调试、数据验证、数据导出等操作，对于做了读写分离的应用，只读访问也使用本账户。
- {user}_mgr：定义为管理账户，拥有多个库下对象操作权限，用于各项目负责人实时操作对象数据。

上述的 3 类用户主要用于应用端的业务，对于 DB 层维护的账户，首先按功能区分，其次在授予权限时也是按照最小粒度权限的原则授予，此外，所有系统维护的账户还会遵循下列两个原则。

- MySQL 数据库的管理员账户改名，不允许出现 root 名称的用户。
- 用户访问域设定为服务器所在 IP 段。

5.4.2　小心历史文件泄密

对特性掌握得越全面，对系统了解得越深入，在执行具体的运维工作时就越能够得心应手，做到有备无患。但是要达到这项要求难度极高、极大，因为向上提升永无极限，我们总能够做得更好。这就要求我们态度上谦虚审慎，同时还要时刻抓紧学习，不忘补充新的知识点。

就以 MySQL 数据库来说，即使当前的密码设置得极为规范，口令保护措施也很到位，但是，如果对 MySQL 的某些特性不了解，就还是有可能留下巨大的系统漏洞。

你们信不信，我反正是信了，执行下列命令看一看吧：

```
$ tail -20 ~/.mysql_history
..............
..............
grant select (phoneno) on jssdb.users to jss_col;
```


```
select * from mysql.columns_priv;
select * from mysql.tables_priv where user='jss_cols';
..............
```

惊讶了吧，前面所做的操作居然都保存在这里，如果一页页去翻开这个文件的内容，你会发现我们执行过的所有操作，包括最初重命名 root 用户和修改 system 系统账户的口令均在其中，嘿嘿，小伙伴儿们当场就震惊了吧！

在 Linux/UNIX 系统下，使用 mysql 命令行工具执行的所有操作，都会被记录到一个名为.mysql_history 的文件中，该文件默认保存在当前用户的根目录下（也可以通过 MYSQL_HISTFILE 环境变量修改保存路径）。

这个设定本意是为了提升 mysql 命令行工具操作体验的，有了它，我们在 mysql 命令行界面就能够方便地使用方向键，上下翻看执行过的命令，但是，又因为它记录了所有执行过的操作，某些情况下又会成为巨大的安全隐患。

基于安全性方面的考虑，对服务端的.mysql_history 文件有必要进行保护，以避免操作被外泄，特别是对于像创建用户、修改密码这类管理操作，如果没有防护，那数据库对某些有心人士来说就是不设防的状态。

如何消除这种隐患呢（也相当于禁用）？有两种方案：

- 一种方案是彻底清除，修改操作系统层的 MYSQL_HISTFILE 环境变量，将其值改为/dev/null，这样所有操作都会被输出到空，操作的历史自然不会被保留。
- 另一个是仍然保留.mysql_history 文件，但是该文件实际上为/dev/null 的软链接，这样所有操作也是会被输出到空，操作的历史不会被保留。

两种方式功能基本相同，原理也一样，只不过一种是在 MySQL 的 DB 层操作，另一种则是在 OS 层实施，究竟选择哪种方式，完全可以由 DBA 自行决定。

这里以第二种方式为例，操作最为简单，在操作系统的命令行界面执行下列命令：

```
$ ln -f -s /dev/null ~/.mysql_history
```

之后再通过 mysql 命令行登录到数据库，方向键上下翻则仅针对当前会话有效，不过退出会话后再次登录，就看不到上次执行过的操作了。

5.4.3　管理员口令丢失怎么办

用户的口令很重要，大家对其重视程度都不低，读过前面小节中的内容后，想必在设置密码时也会花些心思，设定一个不那么有规律的口令。这种设定有时候真是挺矛盾，设置得太简单吧不安全，设置得太复杂又真不容易记，更何况忘记口令这种事儿并不稀奇，看看各大网站的用户登录页面吧，显著位置都留有“找回密码”的链接，就知道这类需求实在太过稀松平常了。这位仁兄，今天，你密码忘了没？

对于数据库软件来说，要是普通用户的口令忘记了倒还好处理，拿系统账户登录进去后简单的一条命令就改了，但是如果系统管理员账户的口令也忘了（这事儿也不稀奇），别担心，照样有方法处理，尽管方法用的非常规，但是：正能量，无所畏。

在操作前必须要首先强调，非常规方式重置系统管理员账户的密码，对 MySQL 服务的正常运行是有影响的，因为操作过程中必须多次重启 MySQL 服务。

MySQL 官方给出的非常规方式重置系统管理员账户有两种方法：

（1）启动 MySQL 服务时附加参数（--init-file），使其执行含有密码重置的脚本，达到修改账户密码的目的。

（2）启动 MySQL 服务时通过附加特殊的参数，使其跳过权限验证，而后登录数据库中重置密码后，再按照正常的方式重启 MySQL 服务。

前一种方式安全性更强，操作的步骤稍多一些，而且不同平台执行的命令也稍有差异，后一种方法通用性强，适用范围广（Windows、Linux、UNIX 平台均适用），因此这里我们重点介绍第二种方式，具体步骤如下：

首先，停止当前的 MySQL 服务，因为没有管理员账户密码，常规关停方法无法使用，只好直接杀进程了，对于 Windows 平台可以在“服务”中停止 MySQL 服务，Linux、UNIX 平台则查找到 mysqld 主进程后杀掉该进程：

```
[mysql@mysqldb01 ~]$ kill `cat /data/mysqldata/3306/mysql.pid`
```

接下来又重新启动 mysqld 服务，启动时在启动命令后附加--skip-grant-tables 选项。该选项的功能正是当有用户连接时跳过检查授权表，直接授予所有登录用户最大权限（相当于所有登录的用户都是系统管理员）。执行 mysqld_safe 命令启动数据库，操作如下（Windows 平台没有 mysqld_safe 脚本，直接执行 mysqld 命令即可）：

```
$ mysqld_safe --defaults-file=/data/mysqldata/3306/my.cnf --skip-grant-tables --skip-networking&
```

大家注意到三思这里还另外指定了--skip-networking 选项，这主要是考虑到附加--skip-grant-tables 选项启动后，连接数据库时不再有权限验证，在此期间如果有其他用户创建连接的话可能存在安全隐患，那么我们可以在启动服务器同时附加--skip-networking 选项，这样该 MySQL 服务不会监听来自 TCP/IP 的连接，相当于禁用了网络上其他主机发出的登录请求，只允许 MySQL 服务本地创建连接，安全性方面更加可靠。

而后就可以无需用户验证，直接登录到 MySQL 数据库服务里：

```
[mysql@mysqldb01 ~]$ mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1
........
........
(root@localhost) [(none)]>
```

所有连接进来的用户现在都是系统用户，默认就是 root，想做啥都可以，对于我们来说，接下来就执行 UPDATE 语句，修改系统管理员账户的密码：

```
(root@localhost) [(none)]> update mysql.user set password=password('5ienet.com') where user='system';
Query OK, 0 rows affected (0.00 sec)
Rows matched: 1  Changed: 0  Warnings: 0
```

关闭 MySQL 服务，这次不用杀进程了，就用常规的 mysqladmin 命令关闭吧：


```
[mysql@mysqldb01 ~]$ mysqladmin shutdown
```

然后按照正常的方式重新启动 MySQL 服务：

```
$ mysqld_safe --defaults-file=/data/mysqldata/3306/my.cnf &
```

系统管理员账户的密码重置完毕，接下来就可以用刚刚设定的新密码来管理你的 MySQL 数据库了。

第 6 章

字符，还有个集

字符集这个东西对于数据库中存储的数据来说非常之重要，特别是对于中文环境这类多字节编码的字符尤其特殊，设置不当就极有可能遇到乱码的情况。基于此，尽管我完全理解大家恨不能马上就进入到 MySQL 数据库中施展拳脚的急迫心情，不过在此之前，我觉着还是有必要先让大家认识 MySQL 中的字符集设定。下面三思简单通过三五个（十/百/千/万）字跟大家阐述一下 MySQL 数据库中的字符集概念和应用。

本章文字内容较多，对于有失眠、多动症、拖延症、强迫症等症状的朋友应有所帮助，欢迎大家对号入座，走过路过不错过。另外，为了更好地促进您中午的睡眠，友情推荐大家可以在午饭后阅读，根据三思本人在营养学方面的造诣，饭后肠胃蠕动所耗费的大量能量，会使得大脑供氧有明显下降，值此大脑昏沉之际研讨枯燥文字，哎呀，想不睡着都难呐！

6.1　基础扫盲

提到**字符集（Character Set）**，我们首先一定要搞明白，“字符集”到底是针对什么，应该怎么理解，我觉着很多人对这个问题模模糊糊的根本原因，实际上就是没能正确理解“字符集”的根本含义。

从最简单的语法上分析，字符集针对的是什么，这个时候一定要注意了，接下来面对的是实际问题，千万不要想得过于复杂，就从最简单的字面意思理解就好。

那么所谓字符集，针对的是“字符”，这样说对不对呢？绝对没有问题。但是也要正确地理解“字符”的概念，提到字符很多人会想什么不是字符呢，“abc”是字符，“123”也是字符，本书中出现的每个符号都是字符。MySQL 中的所谓字符集设置，是针对这些字符吗，我可以负责任地告诉大家，不是，绝对不是。字符集中所谓的“字符”并不是指字符这个形容词，实际上说的是**“字符”类型**。

MySQL 数据库提供了多种数据类型的支持，什么数值型、日期型、二进制类型等都不需要设置字符集，MySQL 数据库中所谓字符集设置，主要是指针对字符类型的设定，比如 char、varchar、text 这类字符型的数据类型。读者朋友们，注意了，本章中当我们再提到字符集时（这里并不仅仅局限于 MySQL 数据库，其他如 Oracle、MSSQL、DB2 等都是同理），如非特别注明，所说的均是指字符类型保存字符的格式。

MySQL 数据库中对字符集设定的支持非常全面，即使相比 Oracle 这种大型数据库软件也毫不逊色，甚至更为灵活，它提供了多种粒度，适应不同的场合和需求，用户可以在服务器、数据库、表甚至列一级进行设置，同时登录到 MySQL 数据库的会话及应用程序连接中也可以进行个性化设定，MySQL 中的 MyISAM、MEMORY 以及 InnoDB 等常用存储引擎均能支持。

大家必须要认识到，字符集并不仅仅对存储的数据有效，在客户端连接服务器端时也与字符集有关。这好像是废话，你想，既然存储的数据有字符集的因素，那么不管客户端准备查询还是保存这些数据，肯定也会与字符集有关系的。话是这么说对吧，但对于很多数据库管理员，他们其实更多是“数据库软件”的管理者，对其中存储的数据介入程度并不深，这种情况下呢，一般默认字符集就能够满足其操作需求。如果环境特殊，不能使用默认的字符

集，MySQL 也提供了相应的方式，可以单独设置当前会话的字符集，后面具体小节中会讲到这一点。

好了，基本情况大家已经都了解了，接下来就让三思多花些笔墨描述，主动帮助大家 kick 到下一层梦镜，好让大家能够睡得更深沉。

6.1.1　关于字符集

先来明确最核心的概念性问题，数据库中的**字符集**究竟是什么？还是问字符集的问题，但是角度有所不同。其实简单讲在数据库看来，字符集就是各种字符编码的一个集合。对于数据库来说，即使是同一个字符，不同的字符集在处理时它的编码格式都有可能不同（废话啊，如果相同，那就没必要搞多种字符集了），那么问题紧跟着就来了，为什么要搞多种字符集呢？此事就说来话长了，不过考虑到咱这本书毕竟不是在做历史考据，太无关的事情扯进来，搞不好就把读者朋友们直接 kick 到 Limbo，那得坐多少站才能回得来啊，所以长话短说吧！

举个例子描述：同样是黑白肤色的大熊猫科动物，搁在大陆叫大熊猫，到台湾就说猫熊，到了美国又改叫 Panda，你要是跑非洲去，没有这种动物，可能都找不出对应的形容词（于是乱码了），对于熊猫来说它自身没发生什么变化，所产生的不同称谓的变化，实际上与地域有很大关联，那么如果把当地拥有的各种词汇集合组成一本字典，对应过来的话，这个字典就是所谓的字符集了（此说并不严谨，本例仅为帮助理解）。

如果要下一个更书面化的定义，那么，**字符集就是指符号和字符编码的集合**。

不同地方的字典当然有可能是不同的，甚至每本字典中的词汇量都不一致，找一本适合的字典非常重要，比如说你给不懂中文的美国朋友看熊猫俩字，它绝对不可能关联到那个毛茸茸的可爱的永远挂着黑眼圈的珍稀动物。

在数据库中应用字符集时，对于具体字符集的设置同样也非常的重要，为了能让字符正确地被保存，同时还能正确地被读取出来，到最终正确地显示给用户，这中间每一个环节都涉及字符集（以及可能发生的转换），只有读和写时，会话所用字符集相互匹配（或者说兼容），最终显示的结果才会正确无误；否则，就会出现不希望看到，但可能又确实常见的现象：乱码。但是，不要怕，只要将本章的内容认真阅读，深入理解，您就可以跟乱码说拜拜。

6.1.2　关于校对规则

MySQL 数据库中提到字符集，有一个关联的关键词是绝对不可被忽略的，那就是**校对规则（Collation）**，官方文档中对此所做的解释如下：顾名思义，校对规则就是指定义的一种比较字符集中字符的规则。也有些资料中将其称为排序规则。

字符集想必大家已经理解，那么校对规则又是怎么一回事儿呢？上面提到的这些概念听起来比较抽象，那么三思还是通过示例来描述吧！应该能够更加清晰一些。

比如我们保存了下列字符到对象的某列中，有“A、B、a、b”四个字符，然后再为上述的每个字符都定义一个数值：A 以 0 表示，B 以 1 表示，a 以 2 表示，b 以 3 表示。

就这个例子来说，A 作为一个符号，与其对应的 0 就是 A 的编码后的形式，上述这 4 个字符以及其编码形式的组合，就是前面所说的**字符集**。那么哪部分指的是校对规则呢？其实脑袋瓜稍灵活一点儿的此时应该也能猜出来，下面就让三思来揭晓谜底吧！

仍以上面的例子来说明，如果我们希望比较多个字符的值，最简单的方式当然就是按照定义好的规则直接对比其编码，按照前面定义的规则，由于 0 比 1 要小，因此我们说 A 比 B 小，应用比较的这个规则，就是所谓的**校对规则**，说得简单点，校对规则的核心就是比较字符编码的方式。

前例中只应用了一项比较的规则，我们将这类最简单的比较所有可能性的校对规则，称

为**二元（Binary）校对规则**。还有些比较复杂的校对规则，比如说大小写等同的规则，在比较时，就需要首先将 a、b 视为等同于 A、B，而后再比较编码（相当于应用了两项规则），这种规则称为**大小写不敏感（Case-insensitive）校对规则**，与之相对应，当然也会有大小写敏感的校对规则，这类规则相对来说要比二元校对规则复杂。

上面所提到的这两种规则，只是字符集的校对规则的最常见形式，实际上排序的规则很多、很复杂，因为我们平常接触到的字符并不仅是 abcd 这类单字节的英文字母，还包括单词、各种符号以及多字节的字符（更加复杂）等，很多的校对规则都拥有一堆的规则，不仅仅是大小写不敏感，还有像汉字中的多音字等。在真正的现实生活中，即使是声称具有高等智慧的人类处理这些时也会晕头转向啊！像三思这种普通话堪称媲美某 AV 台播音员的人物，要是给扔到广东也立马变聋哑，您要是把我扔到美国……亲，我都准备好了，您什么时候扔。

6.2　支持的字符集和校对规则

如前文中所说，字符集有很多很多种，那么，很多初接触 MySQL 数据库的朋友看到这里可能都会想问，MySQL 数据库都支持哪些字符集？这些字符集的校对规则又是什么呢？如何查询当前所使用的字符集？又如何修改当前的字符集呢？不要急，你挨个问，三思可能不会挨个答，因为这些问题有的不是很复杂，有的则很是不简单，我先捡答的上的答！

问：怎么知道数据库当前都支持哪些字符集？

答：可以使用 SHOW CHARACTER SET 语句，例如：

```
(system@localhost) [(none)]> show character set;
+----------+-----------------------------+---------------------+--------+
| Charset  | Description                 | Default collation   | Maxlen |
+----------+-----------------------------+---------------------+--------+
| big5     | Big5 Traditional Chinese    | big5_chinese_ci     |      2 |
| dec8     | DEC West European           | dec8_swedish_ci     |      1 |
| cp850    | DOS West European           | cp850_general_ci    |      1 |
| hp8      | HP West European            | hp8_english_ci      |      1 |
| koi8r    | KOI8-R Relcom Russian       | koi8r_general_ci    |      1 |
| latin1   | cp1252 West European        | latin1_swedish_ci   |      1 |
| latin2   | ISO 8859-2 Central European | latin2_general_ci   |      1 |
| swe7     | 7bit Swedish                | swe7_swedish_ci     |      1 |
| ascii    | US ASCII                    | ascii_general_ci    |      1 |
| ujis     | EUC-JP Japanese             | ujis_japanese_ci    |      3 |
| sjis     | Shift-JIS Japanese          | sjis_japanese_ci    |      2 |
| hebrew   | ISO 8859-8 Hebrew           | hebrew_general_ci   |      1 |
| tis620   | TIS620 Thai                 | tis620_thai_ci      |      1 |
| euckr    | EUC-KR Korean               | euckr_korean_ci     |      2 |
| koi8u    | KOI8-U Ukrainian            | koi8u_general_ci    |      1 |
| gb2312   | GB2312 Simplified Chinese   | gb2312_chinese_ci   |      2 |
| greek    | ISO 8859-7 Greek            | greek_general_ci    |      1 |
| cp1250   | Windows Central European    | cp1250_general_ci   |      1 |
| gbk      | GBK Simplified Chinese      | gbk_chinese_ci      |      2 |
| latin5   | ISO 8859-9 Turkish          | latin5_turkish_ci   |      1 |
| armscii8 | ARMSCII-8 Armenian          | armscii8_general_ci |      1 |
| utf8     | UTF-8 Unicode               | utf8_general_ci     |      3 |
| ucs2     | UCS-2 Unicode               | ucs2_general_ci     |      2 |
| cp866    | DOS Russian                 | cp866_general_ci    |      1 |
| keybcs2  | DOS Kamenicky Czech-Slovak  | keybcs2_general_ci  |      1 |
| macce    | Mac Central European        | macce_general_ci    |      1 |
| macroman | Mac West European           | macroman_general_ci |      1 |
| cp852    | DOS Central European        | cp852_general_ci    |      1 |
| latin7   | ISO 8859-13 Baltic          | latin7_general_ci   |      1 |
```

```
| utf8mb4  | UTF-8 Unicode              | utf8mb4_general_ci  |      4 |
| cp1251   | Windows Cyrillic           | cp1251_general_ci   |      1 |
| utf16    | UTF-16 Unicode             | utf16_general_ci    |      4 |
| cp1256   | Windows Arabic             | cp1256_general_ci   |      1 |
| cp1257   | Windows Baltic             | cp1257_general_ci   |      1 |
| utf32    | UTF-32 Unicode             | utf32_general_ci    |      4 |
| binary   | Binary pseudo charset      | binary              |      1 |
| geostd8  | GEOSTD8 Georgian           | geostd8_general_ci  |      1 |
| cp932    | SJIS for Windows Japanese  | cp932_japanese_ci   |      2 |
| eucjpms  | UJIS for Windows Japanese  | eucjpms_japanese_ci |      3 |
+----------+----------------------------+---------------------+--------+
```

就中文环境来说，最常用的字符集有下面几种：

- GB2312：主要包含简体中文字符及常用符号，这种字符集对于中文字符采用双字节编码的格式，也就是说一个汉字字符在存储时会占两个字节。
- GBK：包括有中、日、韩字符的大字符集，GB2312 也是 GBK 的一个子集，就是说 GB2312 中的所有字符，GBK 中全都有，这种情况下，我们也会将 GBK 称为 GB2312 的超集，GBK 也是双字节编码的格式。将子集中的字符转换成超集中保存不会丢失信息（不会乱码）；但反之则不一定。因此对于超集字符集降级转换成某个子集的操作，需要务必慎重，并反复检验结果，确认不出现丢失字符信息导致乱码的情况。

提 示

此外，还有专门对应繁体中文的 BIG5 字符集，话说 BIG5 也是 GBK 的子集。

- UTF8：它对于英文字符使用一个字节编码，而对于多字节字符（如中文）则使用 3 个字节编码，UTF8 能够支持大部分常见字符，包括西、中、日、韩、法、俄等各种文字，因此可以将上面提到的几种字符集都视为 UTF8 的子集。
- UTF8MB4：前头介绍 UTF8 字符集时，我们知道它能支持西、中、日、韩、法、俄等各种文字，那么接下来要讲的这个字符集就更厉害了，因为它是 UTF8 字符集的超集，UTF8 能够支持的字符，它全都能支持，UTF8 不能支持的字符，它也能支持。这是自 MySQL 5.5 版本才开始新引入的字符集，其引入是为了处理像 emoji 这类表情字符。UTF8MB4 字符集中，一个字符使用 4 个字节编码，能够支持的字符最广，但是相应占用的空间也最大。

问，如何确定某个字符集支持哪些校对规则？

答：一个字符集至少会拥有一个校对规则，显示字符集的校对规则可以使用 SHOW COLLATION 语句，比如说，查看 latin1 字符集所拥有的校对规则：

```
(system@localhost) [(none)]> show collation like 'latin1%';
+-------------------+---------+----+---------+----------+---------+
| Collation         | Charset | Id | Default | Compiled | Sortlen |
+-------------------+---------+----+---------+----------+---------+
| latin1_german1_ci | latin1  |  5 |         | Yes      |       1 |
| latin1_swedish_ci | latin1  |  8 | Yes     | Yes      |       1 |
| latin1_danish_ci  | latin1  | 15 |         | Yes      |       1 |
| latin1_german2_ci | latin1  | 31 |         | Yes      |       2 |
| latin1_bin        | latin1  | 47 |         | Yes      |       1 |
| latin1_general_ci | latin1  | 48 |         | Yes      |       1 |
| latin1_general_cs | latin1  | 49 |         | Yes      |       1 |
| latin1_spanish_ci | latin1  | 94 |         | Yes      |       1 |
+-------------------+---------+----+---------+----------+---------+
```

Default 列显示了校对规则是否是该字符集的默认规则。上述结果中各个校对规则不做详细说明，感兴趣的朋友可以自行参考官方文档等相关资料查阅，不过，MySQL 数据库中字符集的校对规则都有一些共同的特点：

- 每种校对规则只能属于一种字符集，也就是说，不同字符集不可能拥有同一个校对规则。
- 每种字符集都拥有一个默认的校对规则（default collation），SHOW CHARACTER SET 语句显示的结果中，也指明了字符集的默认校对规则。
- 校对规则的名称也有规则，通常开头的字符是校对规则所属的字符集，而后是其所属语言，最后则是校对规则类型的简写形式，有下列 3 种格式：
  - _ci：全称为 case insensitive，表示这是大小写不敏感的规则。
  - _cs：全称为 case sensitive，表示这是大小写敏感的规则。
  - _bin：即 binary，表示这是一个二元校对规则，话说二元校对规则也一定是大小写敏感规则。

接下来，最后一个问题，请问如何查询当前使用的字符集以及如何设置字符集？

答：这个，真不好说。不是这个问题有多难，关键在于 MySQL 此处设定太过于灵活，一句两句说不清楚，下面我们单开一节，详细说说 MySQL 中如何使用字符集和校对规则。

6.3　指定字符集和校对规则

前面提到过，MySQL 数据库能够支持多种字符集，并且每种字符集都拥有一个或多个校对规则，那么为应用端选择一个合适的字符集和校对规则就需要 DBA 把握。当前，选择哪种字符集更好，这个问题咱们暂且放一放，本章重点先说说如何指定吧！

在上一节最后还留下了一个问题：“如何查看当前使用的字符集和校对规则”，为什么说这个问题不好回答呢，就是因为此处 MySQL 数据库的设定呀非常之灵活，当问题不够明确的时候，真是不知道该如何回答。

字符集和校对规则的设定，从大的维度上来说，有两个层次：

- 首先是连接数据库，执行操作时所使用的字符集。
- 其次保存数据时所使用的字符集。

连接时的字符集和保存时的字符集完全有可能不一样，这点倒也容易理解，关键在于不管是连接时，还是保存时，又各有多个粒度，可以精确地为连接时的会话/客户端/操作结果、保存时库/表/列等指定不同的字符集。你看，这种设定够灵活吗？再回头看看问题，当问的不够具体的时候，谁知道亲到底想查看哪个粒度的字符集呀。

因此对于前面的问题，我难以明确回答，不过，换一个角度，本章的内容将从多角度立体式覆盖字符集设定时的方方面面，我想，看完之后，读者朋友们应该能够自己回答这个问题。

6.3.1　服务端设置默认字符集

字符集这么重要的属性，当然不会等到我们实际存储数据时才进行指定。MySQL 服务启动过程中，字符集的设定就已经生效，系统会确定默认所使用的字符集和校对规则，并且，在启动时指定的字符集和校对规则，对应整个 MySQL 服务的全局有效。也就是说，如果没有在更细的粒度对字符集和校对规则进行设置，那么接下来做的所有操作，其字符集的设定都会继承全局粒度所设定的默认字符集和校对规则。

朋友们可能会感到疑惑，我们从安装 MySQL 软件到现在，一步步操作执行下来，印象中似乎没有在何处设置过字符集呀，其实是有的，还记得第 2 章编译 MySQL 软件时指定的两个参数嘛：

```
    -DDEFAULT_CHARSET=utf8 \
    -DDEFAULT_COLLATION=utf8_general_ci \
```

这两个参数决定了本机所运行的 MySQL 服务，在没有做任何其他字符集相关设定的情况下默认时的字符集和校对规则。听仔细了，我说的是**没做其他设定**的情况下，因为，MySQL 中指定字符集的方式太灵活了，即使是全局粒度的设置，方法也不止一种：

- 在编译安装 MySQL（仅限源码编译安装方式）时指定。
- 启动 MySQL 服务时通过参数指定。
- 在参数文件中配置，启动 MySQL 服务时加载参数文件的方式使之生效。
- 在 MySQL 服务运行期间实时修改。

下面逐一介绍。

1．编译 MySQL 软件时指定

正如前面所描述的那样，编译 MySQL 数据库时，就可以通过 default_charset 和 default_collation 两个编译参数，指定默认的字符集和校对规则，如上所示，我们之前设定的字符集为 utf8。而且即使不明确指定，这两个参数也仍然有值，默认情况下将使用 latin1 字符集和 latin1_swedish_ci 校对规则。

2．在启动 MySQL 服务时指定

启动 MySQL 数据库服务时（mysqld 命令或 mysqld_safe 命令）有一堆的参数可以设定，其中与字符集相关的是下面两项系统参数：

- --character_set_server：指定全局粒度的默认字符集。
- --collation_server：指定全局粒度的默认校对规则。

启动时完全没注意过这两个参数？那也没关系，它们是有默认值的，它们的默认值会继承编译 MySQL 软件时 DEFAULT_CHARSET 和 DEFAULT_COLLATION 两个参数所指定的值。噢，你说你管理的 MySQL 数据库并非源码编译安装是吗？那它们的默认值就会是 latin1 字符集和 latin1_swedish_ci 校对规则（并不绝对，某些类型的安装包也可能设定其他字符集）。

3．参数文件中配置

MySQL 服务在启动时，可以通过指定一个参数文件的形式（也就是前面创建的 my.cnf 文件），来简化启动命令行中指定的选项，那么，同样可以将字符集和校对规则的参数设置放在参数文件中。这两项参数放到参数文件中时，参数名与命令行的选项名相同，不过指定参数时不需要"--"字符了，例如：

```
character_set_server=utf8
collation_server=utf8_general_ci
```

下面是道抢答题，我们前面在配置 my.cnf 时有没有指定这两行？为什么？

4．MySQL 服务运行期间修改

前面 3 种方式均可用来指定字符集和校对规则，但若想要变更字符集设置时，动静就太大了，最轻量的变更，想要生效都得重启 MySQL 服务。实际上大可不必如此周折，这两项参数，在 MySQL 服务运行期间作为系统变量存在，而系统变量在 MySQL 服务运行期间，多数都是可以被实时修改的，控制字符集和校对规则的系统变量就属于可被修改那一类。

我们可以通过 SHOW VARIABLES 命令查看当前的系统变量及其变量值，比如说，执行 SHOW VARIABLES 语句查看字符集和校对规则这两项系统变量的值：

```
(system@localhost) [(none)]> show global variables like '%server';
+----------------------+----------------+
| Variable_name        | Value          |
+----------------------+----------------+
| character_set_server | utf8           |
| collation_server     | utf8_general_ci |
+----------------------+----------------+
2 rows in set (0.00 sec)
```

若要将全局粒度默认的字符集改为 gbk，那么执行下列命令即可：

```
(system@localhost) [(none)]> set global character_set_server=gbk;
Query OK, 0 rows affected (0.00 sec)
```

```
(system@localhost) [(none)]> show global variables like '%server';
+----------------------+----------------+
| Variable_name        | Value          |
+----------------------+----------------+
| character_set_server | gbk            |
| collation_server     | gbk_chinese_ci |
+----------------------+----------------+
2 rows in set (0.00 sec)
```

注意看 collation_server 参数的值，它的值也发生了变化，这是因为校对规则是基于字符集的，因此尽管我们没有显式地修改校对规则，但是它的值也自动被修改为所设定的 gbk 字符集的默认校对规则。

也正是基于这一点，在后面小节所做的示例，我们更多地会将演示的重心放在操作字符集上，不会着重介绍“校对规则”方面的内容，因为校对规则是基于字符集的，因此后面段落的内容中，再出现“字符集”这样的字符时，大家也可以认为三思实际要说的是：字符集+校对规则。

提示：全局粒度是什么意思？

在 MySQL 服务运行期间，修改系统变量的值也有作用域的。MySQL 中的系统变量的作用域分为全局（global）和当前会话（session）两类。

对于全局的修改，作用于修改成功后新创建的会话，但对当前执行修改的会话无效，如果是会话级的修改（执行 SET 命令并且未指定 global 选项就是会话级修改），则只作用于当前会话，本次会话结束后，所做修改也自动结束。

此外，需要注意 MySQL 中即使是全局的参数修改，其作用域最多也只在当前 MySQL 服务的生命周期内，MySQL 服务一旦重启，那么之前的设置也全部无效（不管是全局还是会话）。因此要是希望所做的设置永久生效，那么除了在全局粒度修改外，还必须手动修改初始化参数文件，或者是在启动 MySQL 服务时，在命令行中显式指定相关选项值。

6.3.2　连接时指定

客户端连接到 MySQL 数据库服务后所使用的字符集，与 MySQL 服务中设定的若干系统变量有关，前面的小节中也谈到过这方面的内容，就是说默认会继承和使用 MySQL 服务中设置的字符集，也就是全局系统变量 character_set_server 和 collation_server 中指定的字符集和校对规则。

按理说这样不是挺好嘛，所有环节的字符集设置都相同，理论上就不会出现乱码的情况了。理论确实是这样，但实际操作时会面临两方面的挑战。一个是需要它不一致，不要问我怎么会有这种需求，需求从来都是千奇百怪，有时候默认字符集确实不能满足需求（就比如当前字符集设置的是 latin1，但我们实际环境中使用的是 gb2312/gbk 字符集），就是得改，你能怎么样，再说，各个环节的字符集都能被随意设定，这不也是机制灵活的体现嘛！第二，请参考第一点。

总之，确实出现了不一致，那我们能怎么办呢？作为执行者，我们不仅是确保字符被正确地存储，还要确保它能被正确地显示。因此，还是需要多了解机制，搞清楚不同环节对字符集的处理。

字符集并不仅仅只有当存储字符数据时才需要，在客户端与 MySQL 服务器端通信时，字符集同样起着重要的作用，很多情况下，显示的字符出现乱码，也有可能是因为客户端当前的字符集设置与 MySQL 服务器端保存字符时所用的字符集不相符所致。

举一个例子，大家可以先忽略语法，重点看示例中的演示过程，注意看字符集哟。我首先创建一个会话插入一条记录：

```
(system@localhost) [jssdb]> show variables like 'character_set_client';
+----------------------+-------+
| Variable_name        | Value |
+----------------------+-------+
| character_set_client | gbk   |
+----------------------+-------+
1 row in set (0.00 sec)

(system@localhost) [jssdb]> insert into test1 values ('我爱我的祖国');
Query OK, 1 row affected (0.08 sec)

(system@localhost) [jssdb]> select * from test1;
+--------------------+
| v1                 |
+--------------------+
| 我爱我的祖国       |
+--------------------+
1 row in set (0.00 sec)
```

而后在另一个会话中读取这条记录，显示的结果却有所不同：

```
(jss_db@192.168.30.243) [jssdb]> show variables like 'character_set_client';
+----------------------+-------+
| Variable_name        | Value |
+----------------------+-------+
| character_set_client | utf8  |
+----------------------+-------+
1 row in set (0.00 sec)

(jss_db@192.168.30.243) [jssdb]> select * from test1;
+----------------------+
| v1                   |
+----------------------+
| 鎴戠埍鎴戠殑绁栧浗       |
+----------------------+
1 row in set (0.00 sec)
```

我勒个去，乱码啦，怎么办怎么办。别着急，咱们试试对这个客户端所使用的字符集进行设置，执行操作如下：

```
(jss_db@192.168.30.243) [(jssdb)]> set character_set_results=gbk;
Query OK, 0 rows affected (0.00 sec)

(jss_db@192.168.30.243) [(jssdb)]> use jssdb;
Database changed
(jss_db@192.168.30.243) [jssdb]> select * from test1;
+----------------+
| v1             |
+----------------+
| 我爱我的祖国           |
+----------------+
1 row in set (0.01 sec)
```

您瞧，这回字符被正确地显示出来了，而我们所做的不过就是修改了客户端会话中的某个变量值。这就是我们前面所说的，DBAer 不仅只管存储，还得管显示，得一管到底才行。

那么，MySQL 服务是如何响应客户端操作的字符的字符集呢？连接创建后，用户发出的 SQL 语句，MySQL 服务又是如何响应发送查询的结果集或错误信息的字符集呢？

基本上，通过前面这个例子，大家想必也看出来了，还是与系统变量的设定有关。不过，与连接相关的字符集变量可不是 character_set_server 了哟，而是另有参考，并且，还不止一个。客户端信息大致的处理过程如下：

- 客户端发出的 SQL 语句，所使用的字符集由系统变量 **character_set_client** 来指

定。

- MySQL 服务端接收到语句后，会用到 **character_set_connection** 和 collation_connection 两个系统变量中的设置，并且会将客户端发送的语句字符集由 character_set_client 转换到 character_set_connection（除非用户执行语句时，已对字符列明确指定了字符集）。对于语句中指定的字符串的比较或排序，还需要应用 collation_connection 中指定的校对规则处理，而对于语句中指定的列的比较则无关 collation_connection 的设置了，因为对象的表列拥有自己的校对规则，它们拥有更高的优先级。
- MySQL 服务端执行完语句后，会按照 **character_set_results** 系统变量设定的字符集返回结果集（或错误信息）到客户端。

你也许会想看看当前这些系统变量的值，没问题，仍然是使用 SHOW VARIABLES 语句：

```
(system@localhost) [jssdb]> show global variables like 'character_set\_%';
+--------------------------+--------+
| Variable_name            | Value  |
+--------------------------+--------+
| character_set_client     | utf8   |
| character_set_connection | utf8   |
| character_set_database   | utf8   |
| character_set_filesystem | binary |
| character_set_results    | utf8   |
| character_set_server     | gbk    |
| character_set_system     | utf8   |
+--------------------------+--------+
7 rows in set (0.00 sec)
```

这几个系统变量的值，默认继承自服务端启动时默认的字符集设置，也就是我们编译时指定的 utf8。不过你看，有一项例外，那是因为前面我们将 character_set_server 的值设置为了 gbk。

从这个输出结果看到的是全局的设置。当前，每个连接到 MySQL 服务的会话，均可以单独设置连接时的字符集，而且针对这几个系统变量，一般也都只设置会话级的变量值，那么我们再来看一下会话级的系统变量值又是什么呢：

```
(system@localhost) [jssdb]> show variables like 'character_set\_%';
+--------------------------+--------+
| Variable_name            | Value  |
+--------------------------+--------+
| character_set_client     | gbk    |
| character_set_connection | gbk    |
| character_set_database   | utf8   |
| character_set_filesystem | binary |
| character_set_results    | gbk    |
| character_set_server     | utf8   |
| character_set_system     | utf8   |
+--------------------------+--------+
7 rows in set (0.00 sec)
```

标粗的几项变量值被设置为了 gbk，也正是这几项在控制客户端连接和返回信息时，默认使用的字符集。

哟，看起来与全局时的有所不同，怎么弄成这样的呢？在执行 INSERT 语句前，又做了什么呢？接下来，注意了，因为我们就要讲到，如何在创建会话连接后，修改与字符集相关的设定。首先提醒一句，若确实需要修改客户端连接相关的字符集（这种情况很常见），那么并不需要一个个修改这几项系统变量的值，MySQL 提供了专用的方法（还不止一种），可以一次性地修改所有与连接相关的字符集变量设置。

1．SET NAMES

SET NAMES 命令的功能是指定客户端当前会话使用的字符集，语法如下：

```
SET NAMES 'charset_name' [COLLATE 'collation_name']
```

例如，设置当前会话字符集为 utf8，执行命令如下：

```
(system@localhost) [(none)]> set names gbk;
Query OK, 0 rows affected (0.00 sec)

(system@localhost) [(none)]> show variables like 'character%';
+--------------------------+---------------------------------+
| Variable_name            | Value                           |
+--------------------------+---------------------------------+
| character_set_client     | gbk                             |
| character_set_connection | gbk                             |
| character_set_database   | utf8                            |
| character_set_filesystem | binary                          |
| character_set_results    | gbk                             |
| character_set_server     | utf8                            |
| character_set_system     | utf8                            |
| character_sets_dir       | /usr/local/mysql/share/charsets/ |
+--------------------------+---------------------------------+
8 rows in set (0.00 sec)
```

从功能上理解的话，SET NAMES n 相当于同时执行了下列设置：

```
SET character_set_client = n;
SET character_set_results = n;
SET character_set_connection = n;
```

一条命令全搞定，简单且好用，如丝般细致优雅，悠长余韵。

2．SET CHARACTER SET

除了 SET NAMES 命令外，还有 SET CHARACTER SET 命令可以实现类似的功能，该语句的语法也与之类似，具体如下：

```
SET CHARACTER SET 'charset_name'
```

执行 SET CHARACTER SET 命令，相当于设置了下列系统变量：

```
SET character_set_client = x;
SET character_set_results = x;
SET character_set_connection = @@character_set_database;
SET collation_connection = @@collation_database;
```

@@character_set_database 表示全局变量。

下面实际执行看看效果。例如，设置当前会话的字符集为 latin1，执行命令如下：

```
(system@localhost) [(none)]> set character set latin1;
Query OK, 0 rows affected (0.00 sec)

(system@localhost) [(none)]> show variables like 'character_set\_%';
+--------------------------+---------------------------------+
| Variable_name            | Value                           |
+--------------------------+---------------------------------+
| character_set_client     | latin1                          |
| character_set_connection | utf8                            |
| character_set_database   | utf8                            |
| character_set_filesystem | binary                          |
| character_set_results    | latin1                          |
| character_set_server     | utf8                            |
| character_set_system     | utf8                            |
+--------------------------+---------------------------------+
7 rows in set (0.00 sec)
```

提示 1

ucs2、utf16、utf16le 和 utf32 不能被作为客户端字符集使用，也就是说使用 SET NAMES 或 SET CHARACTER SET 命令设置这类字符集无效。

3．固化连接时的字符集设置

正如前面多次提到的，SET NAMES 命令和 SET CHARACTER SET 命令都是基于会话设定的，也就是说，仅作用于当前会话，退出登录后所做设置也就失效了。如果希望设置长期有效，一方面，可以在启动 MySQL 服务时，通过设置相关系统变量，达到永久生效的目的（当然这种方式欠缺灵活性）；另一方面，就是在客户端进行设置，使得与 MySQL 相关的客户端命令在连接时，能够自动使用我们设定好的字符集，避免每次在会话操作前再单独设定。

MySQL 数据库的客户端程序，包括 mysql、mysqladmin、mysqldump、mysqlimport 等命令行工具，都是按照下列规则，读取连接时的默认字符集设定，优先级也是依次递增：

- 默认情况下，使用编译时指定的默认字符集，在本环境中当然就是 utf8 了。
- 程序能够自动检查当前操作系统环境变量中设置的字符集，比如像本地操作系统环境变量 LANG 或 LC_ALL 设置的语言，因此用户也可以配置操作系统的环境变量，来修改客户端连接后的默认字符集，但是这里需要注意的一点是，一般来说操作系统能够支持的字符集要比 MySQL 多，极有可能 OS 层指定的字符集在 DB 层没有对应，这种情况下，MySQL 仍然会设置编译时指定的字符集为默认字符集了。
- 对于支持 default_character_set 选项的命令行工具（常用的命令全都能够支持），可以通过该参数设置连接后的默认字符集。为了简化操作，甚至可以将"default_character_set"系统变量放在 my.cnf 文件的[mysql]或[client]区块，这样当 mysql 客户端连接到服务器后，它就能按照前面指定的字符集自动设置 character_set_client、character_set_results 以及 character_set_connection 几个系统变量。

比如说，我们当前有一套网站系统，号称面向全球用户提供服务，当然啦，实际上主要用户还是以中文群体为主，不过为了保证字符能被正确地识别和存储，默认字符集还是需要设置为 utf8（或 utf8mb4），以支持更多的语言，而不仅仅只是中文。可是作为维护者，由于我们的日常操作环境仍是中文占主导，平时更新或维护的数据也都是中文字符，因此在操作时，就需要字符集保持为 gb2312 或 gbk，我们又不希望每次连接到 MySQL 服务后都通过 SET NAMES 进行修改，那么，就可以考虑在参数文件 my.cnf 中的[mysql]区块，增加一行：

```
[mysql]
default-character-set=gbk
```

这样，只要我们使用 mysql 命令行工具连接到服务器后，连接的默认字符集（即 character_set_client、character_set_results 和 character_set_connection 几个系统变量）就都会是设定好的 GBK 字符集了。

不过，必须意识到的一个现状是，说到底，DBA 再优秀，一个人的力量毕竟有限，不可能他所接触的每台服务器都由他安装配置，或者按照他的风格去设置。尽管前面提到的一些方法配置得当时，用于提高工作效率着实有效，但是，就我看来，养成一个良好的习惯更重要，只有把正确的习惯深深地烙印在脑海中，形成操作 DB 时的本能，才能更好地适应环境，在实际工作中，不管应对什么样的工作环境，才能够确保所做操作安全、可靠，才能赢得同事和领导对你的信任和肯定。

前面说的这段话您听明白什么意思了吗？确实有点儿隐晦，好吧，那我换个说法，涉及字符操作时，先执行 SHOW VARIABLES LIKE 'charact%'看一看当前的字符集到底是什么吧，不要因为想当然而导致操作数据出现错误。

6.3.3　保存时指定

连接时指定的字符集是确保在交互过程中，字符数据被正确编码，此外，作为数据库的主要功能——存储，更要确保数据在保存时也使用正确的字符集。

在 MySQL 数据库中，提供 4 种不同的粒度，用以指定存储时数据使用的字符集：

（1）server，全局级，本领最大就是 only you，这个在前面 6.3.1 节已经说得够多了。

（2）database，数据库级。

（3）table，表级。

（4）column，列级。

这 4 种级别作用域依次递减，不过优先级是依次递增。怎么理解呢？举例来说，当在 server 级别指定字符集后，如果在 database/table/column 级未设置字符集，那么默认就会继承 server 级别的设定，不过如果在低一级别中明确指定，比如创建表时明确指定了字符集，那么该表的字符集又会以表级指定的为准。

看起来很灵活是吧！这里又不得不再次提及三思的这种观点：灵活意味着可以很复杂。你看看，任意一个级别都可以设定不同的字符集和校对规则设定，你倒是说说，要是被人问起“当前”设置是什么，该怎么回答。这个到底说的是服务器呢，还是数据库呢，这还没算上用户对当前连接的会话进行的字符集设置呢。正是因为有如此灵活的设定，才在面对前面小节最后提出的问题时难以给出简明的回答，因为在没有明确“当前”这个字符定义的情况下，实在无法确定查询哪个参数的哪个值。

当然啦，上面这段描述只是为了帮读者朋友加深印象，千万不要产生畏难情绪，这个情况在三思看来并不算复杂，它只是灵活，非常的灵活，因此在介绍本小节内容时会尽可能多通过示例来说明，示例说明不了的，三思尽可能就不说了。

1．*在数据库级指定*

每个 MySQL 服务中的数据库都可以单独设置字符集和校对规则，这种设置既可以在创建数据库时指定，也可以在创建之后通过 ALTER DATABASE 语句进行修改，指定字符集和校对规则的语法特别简单，具体如下：

```
[[DEFAULT] CHARACTER SET charset_name]
[[DEFAULT] COLLATE collation_name]
```

其中[DEFAULT]关键字是可选项，就实际效果来说，指定或不指定都没有区别，大家直接忽略它也行。

另外需要提示一点的就是，在 MySQL 中指定字符集，完整句法是“CHARACTER SET n”，这个写法也可以简化为“**CHARSET** n”，功能完全相同，后者可视作前者的同义词。我通常都是使用后一种写法，不要小看这少敲的几个字符，累积起来能提高不小的效率呐，你们信不信，我说的我自己都快信了。

下面，我们创建一个名为“5ienet”的数据库，并指定该库的默认字符集为 latin1，执行语句如下：

```
(system@localhost) [(none)]> create database 5ienet charset latin1;
Query OK, 1 row affected (0.00 sec)
```

修改数据库的字符集也非常简单，比如说修改 5ienet 库的字符集为 utf8，执行命令如下：

```
(system@localhost) [(none)]> alter database 5ienet charset utf8;
Query OK, 1 row affected (0.00 sec)
```

前面执行了两项操作，不管是创建还是修改，都只指定了字符集，而没有指定校对规则，这是因为字符集和校对规则都是可选参数，大家也可以只指定校对规则而不指定字符集，或者两个都指定，甚至两个都不指定也行，它就继承全局粒度设定的默认值呗。

不过，考虑到字符集和校对规则两个都可选的情况下，实际应用时可能存在多种情况，下面简要说一说。一般来讲，MySQL 会按照下列规则来设置默认的字符集和校对规则：

- 如果同时指定 CHARACTER SET 和 COLLATE 选项，那没说的，就按指定的值处理。
- 如果仅指定了 CHARACTER SET，那么 COLLATE 会继承指定字符集的默认校对规则（还记得如何查看字符集的默认校对规则吗？要是忘记了就再回去看看 6.2 节吧）。

- 如果仅指定了 COLLATE 选项，那么 COLLATE 所属的字符集就是该数据库的默认字符集了。
- 如果两参数均未指定，那么就是前面所说的，该数据库将会继承全局粒度所设定的字符集和校对规则，不过继承并不是说直接应用 character_set_server，而是另有变量：character_set_database 和 collation_database。

数据库级的字符集设置，是保存在数据库同名的操作系统目录下的 db.opt 文件中：

```
$ more /data/mysqldata/3306/data/5ienet/db.opt
default-character-set=utf8
default-collation=utf8_general_ci
```

直接修改该文件也可以达到修改库级字符集设定的目的，但是一般不需要这么干，而且操作系统层修改并不是实时生效的，需要重启 MySQL 服务才能看到。

要在 MySQL 中查看某数据库的字符集和校对规则，最简单的方式就是直接 SHOW 它：

```
(system@localhost) [(none)]> show create database 5ienet;
+----------+-----------------------------------------------------------------+
| Database | Create Database                                                 |
+----------+-----------------------------------------------------------------+
| 5ienet   | CREATE DATABASE `5ienet` /*!40100 DEFAULT CHARACTER SET utf8 */ |
+----------+-----------------------------------------------------------------+
1 row in set (0.00 sec)
```

数据库的字符集作用域包括在该库下创建的表和列，因此，该库下所有表默认均会继承该库所设置的字符集，甚至于使用 LOAD DATA INFILE 向该库下对象加载数据，如无明确指定，默认也会继承库一级的字符集。

2．在表级指定

在创建表对象时，可以明确地给表对象指定字符集和校对规则，如果没有指定的话，它就会继承所在数据库粒度设定的字符集。建表或修改表时与字符集相关的语法，和在数据库级操作的语法完全相同：

```
[[DEFAULT] CHARACTER SET charset_name]
[[DEFAULT] COLLATE collation_name]
```

下面创建一个名为 t1 的表对象，先不指定字符集，执行语句如下：

```
(system@localhost) [5ienet]> create table t1 (id int);
Query OK, 0 rows affected (0.00 sec)
```

再创建一个名为 t2 的表对象，指定该表默认字符集为 latin1，执行语句如下：

```
(system@localhost) [5ienet]> create table t2 (id int) charset latin1;
Query OK, 0 rows affected (0.01 sec)
```

分别来查看这两个对象的字符集：

```
(system@localhost) [5ienet]> show create table t1;
+-------+---------------------------------------+
| Table | Create Table                          |
+-------+---------------------------------------+
| t1    | CREATE TABLE `t1` (
  `id` int(11) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8 |
+-------+---------------------------------------+
1 row in set (0.00 sec)

(system@localhost) [5ienet]> show create table t2;
+-------+---------------------------------------+
| Table | Create Table                          |
+-------+---------------------------------------+
| t2    | CREATE TABLE `t2` (
  `id` int(11) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1 |
+-------+---------------------------------------+
1 row in set (0.00 sec)
```

如结果中所示，当创建对象时，没有指定字符集，那么它就会继承数据库粒度的字符集，如果明确指定了字符集，那么该表对象的字符集就是我们指定的字符集。

这里列出的两个示例同样没有涉及校对规则，在表粒度的校对规则继承与数据库粒度时基本相同，唯一的区别是数据库粒度时字符集是继承自全局粒度，而在表粒度默认的字符集和校对规则继承自数据库粒度的设置。

3．在列级指定

所有**字符类型**的列（即列的数据类型为 CHAR/VARCHAR 或 TEXT，另外 ENUM 和 SET 类型也适用）均可以在创建时指定字符集和校对规则，当然也可以通过 ALTER TABLE 命令对字符集进行修改。在通过 ALTER TABLE 语句定义列类型时，指定字符集和校对规则的语法如下：

```
col_name {CHAR | VARCHAR | TEXT} (col_length)
    [CHARACTER SET charset_name]
    [COLLATE collation_name]
col_name {ENUM | SET} (val_list)
    [CHARACTER SET charset_name]
    [COLLATE collation_name]
```

从语法上大家应该也看出来了，跟前面讲过的库级粒度和表级粒度一样，只不过列级的粒度最小，它能直接定义某个字符类型列的字符集。具体处理时，列粒度的字符集和校对规则的处理跟前面库级和表级也都差不多，唯一的差别就是，列粒度的字符集和校对规则默认会继承表级粒度的设置。

列粒度的知识点跟前面的内容重合度很高，这里既不重复举例也不准备多谈了，本小段前面的内容大家三两眼就能看完，接下来我想趁此时机简单说一下修改字符集设置对已有数据的影响。

前面讲过存储相关的字符集粒度分为 4 级：全局、库、表、列。可以说，全局和库级粒度的字符级设置可任意修改，它们不会对现有数据造成影响，最多所能影响到的也是修改设置后新增的数据。不过对于表粒度和列粒度中字符集的修改就需要慎重处理了，因为表和列中真正保存着数据，对现有数据的字符集进行变更，如果操作不慎，是会**丢失**数据的。

当我们执行 ALTER TABLE 语句，修改表对象或表中某个字符类型列的字符集时，MySQL 都要尝试，将字符从原有字符集转换成新指定的字符集进行编码保存，在这个过程中，如果字符集之间不兼容，就会丢失数据。这部分数据如果是在转换过程中丢失的，那么就没有回滚的可能，也就是说在没有备份的情况下，丢失的数据就永远丢失了，而不是说你再把字符集修改回原样，它就还能回去。

怎么，你想问什么情况下会丢失数据，这个细说起来就复杂了，简单概括就两句话：子集到超集的转换没有问题，但超级转换成子集会有问题。

举例来说，latin1 字符集中的西方字符在 gb2312 字符集中都有对应（支持），那么将字符集从 latin1 转换到 gb2312 就是安全的，同理，gb2312 到 gbk 也是安全的，gbk 到 utf8 也是安全的，当然 latin1 到 utf8 更是安全的了。

反之，utf8 转换到 gbk 就**不一定**，比如说 utf8 字符集中能够支持俄文字符，但是 gbk 不支持呀，这种情况下，将 utf8 字符集的表或列转换成 gbk 字符集，就有可能会丢失数据。可是既然并不能完全支持，为什么这里又说有可能呢，这因为尽管不是完全支持，但必须还是有能够支持的字符嘛，比如表或列的字符集尽管是 utf8，但是如果其中存储的都是 gbk 能够支持的中文字符，并没有不被支持的字符，这种情况超集到子集的转换也是安全的。

6.4　字符集操作示例

前面说了不少与字符集相关的内容，尽管也有示例辅助理解，但还是不够形象、具体，下面三思想通过一个最简单的示例：“向某测试表插入记录，确保写入的记录能被正确地保存，并能被正常地读取”。这是再正常不过的需求，三思尽可能使这个示例保持简单，并在操作过程中，为大家演示 MySQL 连接和存储这两个层面和多个粒度中，字符的不同字符集应用和实际效果。

先来创建一个测试表 t3，拥有 3 个列，并为每个列分别指定不同的字符集，所指定的 3 种字符集，也是目前 MySQL 环境最为常见的字符集，执行操作如下：

```
(system@localhost) [5ienet]> create table t3 (
    -> v1 varchar(20) charset latin1,
    -> v2 varchar(20) charset gbk,
    -> v3 varchar(20) charset utf8);
Query OK, 0 rows affected (0.01 sec)
```

设置当前客户端连接会话的字符集为 utf8：

```
(system@localhost) [5ienet]> set names utf8;
Query OK, 0 rows affected (0.00 sec)

(system@localhost) [5ienet]> show variables like 'character%';
+--------------------------+----------------------------------+
| Variable_name            | Value                            |
+--------------------------+----------------------------------+
| character_set_client     | utf8                             |
| character_set_connection | utf8                             |
| character_set_database   | utf8                             |
| character_set_filesystem | binary                           |
| character_set_results    | utf8                             |
.....................
```

向这个表中插入一条记录，每个列都指定相同的值，中英文字符混合：

```
(system@localhost) [5ienet]> insert into t3 values ('cn 中国','cn 中国','cn 中国');
ERROR 1366 (HY000): Incorrect string value: '\xD6\xD0\xB9\xFA' for column 'v1' at row 1
```

返回 1366 号错误信息，细看提示就能明白，原来在提示 v1 列指定了错误的字符串值，导致插入失败。这个错误提示可以理解，因为前面说过，latin1 是西文字符集，并不支持中文这样的多字节字符，因此插入时就报错了。

不过，考虑到 MySQL 5.6 正式版本刚出不久，大部分用户仍在使用 5.6 之前的版本，需要提醒大家的是，在 MySQL 5.6 之前版本，1366 号信息并非错误，而是一条警告，也就是说在之前版本中，出现这种情况只会抛出一条警告，但插入操作仍能成功。但是，都出现警告了，实际插入的数据出现异常的概率也就相当的高了，若是以为插入成功，但查询时才发现信息错误，那就麻烦了。因此这就需要大家更加慎重地对待，因为有时候操作失败报错，要比操作（同样）失败报警更合理一些呐。进入 5.6 版本后，MySQL 修改 1366 号的错误级别，想必也是认识到了这一点，我个人对这类改动非常认同。

提示：此处 1366 错误级别由系统变量 sql_mode 来控制

sql_mode 系统变量，顾名思义用来控制 SQL 模式，MySQL 提供的模式众多，不同模式提供了不同的功能或限制条件。在 MySQL 5.6 版本之前，sql_mode 默认为空，而进入到 5.6 版本后，sql_mode 默认值改为 STRICT_TRANS_TABLES，这个变量值的功能是对于支持事务的存储引擎对象，启动严格限制模式，这种情况下，就会出现插入值非法则直接报错，而非警告。

若仍然希望像 5.6 版本之前，插入字符串值不匹配抛出警告而非错误，则将 sql_mode 值改为空，如果该系统变量拥有多个值，则去除 STRICT_TRANS_TABLES 值即可。

下面修改一下 col1 待插入的值，而后再次执行 INSERT 语句：

```
(system@localhost) [5ienet]> insert into t3 values ('china','cn 中国','cn 中国');
ERROR 1366 (HY000): Incorrect string value: '\xAD\xE5\x9B\xBD' for column 'v3' at row 1
```

又报了几乎一模一样的错误，不过这回变成 v2 列字符串，即使我们继续尝试，把 v2 列值改一下，再次尝试插入时又会抛出错误，提示 v3 列的字符有误。

您说像 v1 这种 latin1 编码的列，插入中文字符出现错误还可以理解，因为它确实不支持中文编码，但为什么 v2 和 v3 这两列也会是乱码呢？这两个一个使用 gbk 编码，一个基于 utf8 编码，应该是都能够支持中文字符的呀！

既然存储阶段的字符集设置没有问题，我们输入的字符也没有问题，那就只有一个可能：连接阶段的字符集编码不对。可是，回想一下，在操作之前，我们已经显式地指定了当前连接会话的字符集为 utf8 了呀，utf8 能够支持中文的不是嘛？

你看，这就是一处新手极容易陷入的理解误区，utf8 编码能够支持中文不假，但是，谁说 utf8 编码格式下的“中国”，就是我们输入的“中国”这两个字符呢？这明明是 gb2312 或者说 gbk 编码的字符形式。

插入时报错，不是我们输入的不对，或者表列的字符集设置不对，而是客户端的字符集和存储所用字符集不匹配的结果。您若还没想通，咱们通过实际操作验证一下，先把当前客户端会话的字符集改为 gbk，然后再执行相同的 INSERT 操作看一看：

```
(system@localhost) [5ienet]> set names gbk;
Query OK, 0 rows affected (0.00 sec)

(system@localhost) [5ienet]> insert into t3 values ('china','cn 中国','cn 中国');
Query OK, 1 row affected (0.03 sec)
```

语句居然成功执行。

查询一下 t3 表对象中的数据，刚刚插入的记录是否能正常显示，执行 SELECT 语句：

```
(system@localhost) [5ienet]> select * from t3;
+-------+--------+--------+
| v1    | v2     | v3     |
+-------+--------+--------+
| china | cn 中国    | cn 中国    |
+-------+--------+--------+
1 row in set (0.00 sec)
```

v1 列不说了，单字节的西文字符想乱码很难的。包含中文的 v2 和 v3 两列的显示结果也完全正常。这就是我们在 6.3 节中反复说的，连接时字符集和存储时的字符集一致，结果才能正常显示。

您是否想问 utf8 字符集的 v3 列为什么也能正常显示？这可不就是 character_result 系统变量的功劳了嘛，它被转换成了 gbk 字符集形式嘛。

而后，我们再把连接会话的字符集改为 utf8，看看表中的记录又会变成什么样呢：

```
(system@localhost) [5ienet]> set names utf8;
Query OK, 0 rows affected (0.00 sec)

(system@localhost) [5ienet]> select v1,v2,v3,length(v1),length(v2),length(v3) from t3;
+-------+----------+----------+-----------+-----------+-----------+
| v1    | v2       | v3       | length(v1) | length(v2) | length(v3) |
+-------+----------+----------+-----------+-----------+-----------+
| china | cn 涓  浗  | cn 涓  浗  |          5 |          6 |          8 |
+-------+----------+----------+-----------+-----------+-----------+
1 row in set (0.00 sec)
```

大家首先注意看每列值的长度（跟你想的一样不？如果不一样，再想想能想明白不？如果想不明白，知道我邮件地址不？如果不知道……呃，太好了）。

而后再来看 v2、v3 两列的列值，字符倒是显示出来了，可是这回显示的字符比较怪。难道使用 utf8 字符集时，“中国”的编码会是“涓 浗”这种都不知道怎么读的字符形式，让我

们试一试吧，实践出真知，怎么试呢？方法太多了，我们选个复杂的，INSERT 看看呢：

```
(system@localhost) [5ienet]> insert into t3 values ('cn','cn涓　浗','cn涓　浗');
Query OK, 1 row affected (0.03 sec)
```

修改客户端会话的字符集为 gbk，再次查询 t3 表中的记录，看看刚刚插入记录中的字符能否正常显示：

```
(system@localhost) [5ienet]> set names gbk;
Query OK, 0 rows affected (0.00 sec)

(system@localhost) [5ienet]> select * from t3;
+-------+--------+--------+
| v1    | v2     | v3     |
+-------+--------+--------+
| china | cn中国    | cn中国    |
| cn    | cn中国    | cn中国    |
+-------+--------+--------+
2 rows in set (0.00 sec)
```

结果告诉我们，记录没有任何问题。看来，UTF8 编码中的“涓　浗”正是 GBK 编码中的“中国”，我想看到这里，大家应该能够明白，为何我们第一次、第二次及第三次的 INSERT 会失败，以及后续的查询结果中，字符变来变去的原因了吧！

现在我想把 v3 列的字符集转换成 gb2312，大家说能行吗：

```
(system@localhost) [5ienet]> alter table t3 modify v3 varchar(20) charset gb2312;
Query OK, 2 rows affected (0.02 sec)
Records: 2  Duplicates: 0  Warnings: 0
```

改当然能改，ALTER TABLE 的语法只要没敲错，执行是没问题的，但是修改后列中存储的数据是否仍然正确，就要看字符集的转换过程中是否有不兼容的情况发生。

就我们这个例子，大家说会有不兼容吗？不会，“cn 中国”这个字符在 gb2312 字符集中完全能够正确编码，不会丢失数据。我这个论断可是在没执行查询之前给出的，大家要是也都像我这么自信，那对字符集的理解应该就没问题了，可以跳过本章，阅读下面的内容了：

```
(system@localhost) [5ienet]> select * from t3;
+-------+--------+--------+
| v1    | v2     | v3     |
+-------+--------+--------+
| china | cn中国    | cn中国    |
| cn    | cn中国    | cn中国    |
+-------+--------+--------+
2 rows in set (0.00 sec)
```

您还想继续往下看是吗，好吧，再来一个问题，要是我把 v3 列的字符集改为 latin1，您说数据能正常操作不，别看结果，先自己给个答案，然后跟结果对比一下：

```
(system@localhost) [5ienet]> alter table t3 modify v3 varchar(20) charset latin1;
ERROR 1366 (HY000): Incorrect string value: '\xD6\xD0\xB9\xFA' for column 'v3' at row 1
```

答对了没，要是答对了就继续往下看，没答对就翻到本章第一页从头看起！

6.5　角落里的字符集设置

有人的地方，就有江湖，有字符的地方，就有字符集。在 DB 层的角落，隐藏着一群与我们的操作息息相关的提示信息，它们默默地工作呢，并且被我们理所当然的忽略了，其实它们也有自己的空间，它们也有字符集。

6.5.1　字符串的字符集

前面我们提到，字符集是针对 MySQL 数据库中的字符类型，我们一般设置字符集，都是对存储对象进行设置，这里还有一个很细节的问题，不知道大家有没有想到过，对于那些并非保存在数据库对象里的字符是否拥有字符集呢？比如下面这个语句：

```
Select 'cn 中国';
```

这个语句中的字符串有字符集吗？这点毋庸置疑，必须有啊，字符类型都有字符集。不过出现在 SQL 语句中某个角落的字符串，多数情况下我们都是理所当然地就用了，它们的字符集却被我们忽视。

前面小节中的示例跟这个也有关联，执行的 INSERT 语句，为什么有时候读出来会是乱码，可不就是因为输出的字符，其默认字符集与保存时字符集不匹配嘛，因此当我们通过 SET NAMES 命令修改了字符集后，操作结果就能够正常显示。所以你看，每个字符串都有对应的字符集和校对规则，不管它从哪儿来，在哪儿出现。

MySQL 数据库在字符集的设置方面想得比较全面，针对字符串，用户可以在使用时，通过相应语句来显式地设置字符集及校对规则，其基础语法如下：

```
[_charset_name]'string' [COLLATE collation_name]
```

这其中：

- [_charset_name]：指定字符集，其中_符号是固定格式，后面跟字符集名称，由于是可选项，因此一般都忽略了。
- 'string'：列或列值。
- [collation_name]：指定校对规则。

有了“_charset_name”这样的表达式，就应该想到，SET NAMES 其实也并不是必需的，不管做什么操作，我们的核心目标都是希望输入输出时的字符集兼容匹配，这也是字符数据得以正确显示的前提。

例如，紧接前面的示例，下面 SQL 语句中选择的几个字符串的列值结果是相同的：

```
(system@localhost) [5ienet]> SELECT 'cn 中国', _gbk'cn 中国', _utf8'cn 涓  涤' COLLATE utf8_general_ci;
+--------+--------+-----------------------------------------+
| cn 中国 | cn 中国 | _utf8'cn 涓?涤' COLLATE utf8_general_ci |
+--------+--------+-----------------------------------------+
| cn 中国 | cn 中国 | cn 中国                                   |
+--------+--------+-----------------------------------------+
1 row in set (0.00 sec)
```

这就是_charset_name 表达式的作用，它会告诉解析器使用其后的字符串作为字符集。

大多数情况下，我们都忽略了指定字符串的字符集，不管指定不指定，字符串都是有字符集的，那么 MySQL 是采用何种策略确定该字符串的字符集呢？这其中有一定的规则，取决于当前的系统环境设置：

- 如果同时指定了字符集和校对规则，则按照指定的设置。
- 如果仅指定了校对规则，那么采用校对规则所属的字符集作为默认字符集。
- 如果均未指定（通常都是这样），那么会按照系统变量 character_set_connection 和 collation_connection 的设置作为默认的字符集和校对规则。

6.5.2　错误提示的字符集

服务端返回的错误或警告信息也是字符，当然也有字符集，前面的示例操作时就碰到过一些警告信息，大家想必也注意到返回的信息跟我们想象中的不太一样，本小节就跟大家说道说道 MySQL 在这方面的设计。

对于 MySQL 服务来说，服务端固定使用 utf8 字符集组织错误信息，返回到客户端时，转换成 character_set_results 系统变量指定的字符集。

服务端在组织错误或警告信息时，按照下列的方式处理信息的几个组成部分：

- 首先消息模板使用 utf8 字符集。
- 而后，消息模板中的参数替换成指定的错误事件，包括几个子项：
  - 标识符，比如表名或者列名这类内部即使用 utf8 字符集的仍然用该字符集输出。

- 字符串值（不含二进制）则从原始字符集转换成 utf8。
- 二进制字符串转换成字节方式表示，范围在 0x20～0x7e 之间，超出该范围则使用\x 十六进制编码。比如说一个键复制错误，尝试插入 0x41CF9F 到 VARBINARY 唯一列时，返回的错误信息如下：

```
Duplicate entry 'A\xC3\x9F' for key 1
```

这些信息被组合后返回消息到客户端，服务器将其从 utf8 转换成 character_set_results 系统变量所指定的字符集，如果 character_set_results 变量值为空或 binary，那么就不会发生转换操作，当然如果该变量设置的值就是 utf8 也不会发生转换。

如果信息中的字符串，不能被以 character_set_results 变量中所指定的字符集展示，那么转换过程中可能就会触发另外的编码方案：

- 字符在基本多文种平面（Basic Multilingual Plane、BMP，也即 Unicode 编码的 0 号平面）范围 0x0000～0xFFFF 区间的使用"\nnnn"标记输出。
- 字符不在 BMP 范围 0x01000～0x10FFFF 区间的使用"\+nnnnnn"标识输出。

设置错误信息的语言

默认情况下，mysqld 进程会使用英文返回错误信息，这当然不代表它只能以英文显示，实现上它还支持其他很多种语言，包括 Czech、Danish、Dutch、Estonian、French、German、Greek、Hungarian、Italian、Japanese、Korean、Norwegian、Norwegian-ny、Polish、Portuguese、Romanian、Russian、Slovak、Spanish 或者 Swedish，用户可以选择上面提到的任意一种语言来显示错误信息，遗憾的是目前还不支持中文。

有的朋友可能又有问题了，如何设置错误信息的显示语言呢？这个问题不复杂，不过咱们得从头说起。MySQL 服务在启动时，会读取--lc_messages_dir 选项的值，并会在该选项指定的路径下查找错误信息对应的语言文件。该选项的默认路径是 MySQL 软件安装目录下的 share 目录，本环境中，就是在/usr/local/mysql/share/目录下：

```
[root@mysqldb01 ~]# ll /usr/local/mysql/share/
total 1408
drwxr-xr-x. 2 mysql mysql   4096 Mar  6 09:27 aclocal
drwxr-xr-x. 2 mysql mysql   4096 Mar  6 09:27 bulgarian
drwxr-xr-x. 2 mysql mysql   4096 Mar  6 09:27 charsets
drwxr-xr-x. 2 mysql mysql   4096 Mar  6 09:27 czech
drwxr-xr-x. 2 mysql mysql   4096 Mar  6 09:27 danish
-rw-r--r--. 1 mysql mysql  25575 Jan 23 00:54 dictionary.txt
drwxr-xr-x. 2 mysql mysql   4096 Mar  6 09:27 dutch
drwxr-xr-x. 2 mysql mysql   4096 Mar  6 09:27 english
-rw-r--r--. 1 mysql mysql 501829 Jan 23 00:54 errmsg-utf8.txt
.............
.............
drwxr-xr-x. 2 mysql mysql   4096 Mar  6 09:27 spanish
drwxr-xr-x. 2 mysql mysql   4096 Mar  6 09:27 swedish
drwxr-xr-x. 2 mysql mysql   4096 Mar  6 09:27 ukrainian
```

找到错误消息对应语言的目录了，接下来怎么办呢，它还有一个选项--lc_messages，这个选项就是用来指定错误信息的语言。

lc_messages_dir 是个只读的系统变量，MySQL 服务启动后就无法修改，不过这没关系，它只是指定路径而已，决定语言的 lc_messages 系统变量则可以在 MySQL 服务运行时随意修改，并且既可以在全局粒度修改，也可以在会话粒度修改。

下面咱们就试一下，修改当前错误信息的语言为法文，即修改当前会话中 lc_messages 系统变量的值，执行命令如下：

```
(system@localhost) [5ienet]> set lc_messages=fr_FR;
Query OK, 0 rows affected (0.00 sec)
```

触发个错误看看：

```
(system@localhost) [5ienet]> select abc;
```

```
ERROR 1054 (42S22): Champ 'abc' inconnu dans field list
```

懂法文的给翻译翻译，这说的到底是嘛意思呐。

提示一点，本节提到的两个系统变量 lc_messages_dir 和 lc_messages 是从 MySQL 5.5 版本才开始引入的，在之前的版本中，要设置语言是通过--language 选项，该参数相当于—lc_messages_dir 和—lc_messages 的集合，比如说，仍然设置法语显示，则在 MySQL 5.5 之前的版本中，需要在启动时加载--language 选项，执行命令如下：

```
$ mysqld_safe --language=/usr/local/mysql55/share/french
```

6.5.3　国家字符集

熟悉 MySQL 数据类型的朋友都知道，MySQL 中还存在 NCHAR、NVARCHAR 这样的字符类型，这其中的 N 代表的就是 NATIONAL，这类字符类型在 MySQL 数据库中拥有固定的字符集设置，在 MySQL 5.6 版本中，使用 utf8 作为这种类型在存储时的字符集。

也就是说，在 MySQL 5.6 版本中，不管如何设置字符集，当使用 N[char/varchar/text]数据类型时，这些列的字符集均为 utf8。这也是它被称为“国家字符集”的原因，拥有更好的兼容性，不管所使用的字符究竟是何种，均能够被正确支持和保存。

基于这一点，下面几种列的字义在功能上是完全相同的：

```
CHAR(10) CHARACTER SET utf8
NATIONAL CHARACTER(10)
NCHAR(10)
```

在通过 SQL 语句操作字符串时，也可以简化的形式创建字符类型为国家字符集，例如：

```
(system@localhost) [5ienet]> select N'涓　浗', n'涓　浗', _utf8'涓　浗';
+------+------+------+
| 中国 | 中国 | 中国 |
+------+------+------+
| 中国 | 中国 | 中国 |
+------+------+------+
1 row in set (0.00 sec)
```

关于字符集的内容还有很多，不仅仅是指某些设置上的细节，而是说 MySQL 数据库提供很丰富的与字符集相关的功能。前面的段落尽管也花了不少篇幅，但其实只是浅显地向大家表明了 MySQL 中常见的字符集处理方法。

其他还包括时区啦、元数据啦、UNICODE 编码支持啦等与字符集相关的内容，我们甚至可以自己增加新的字符集或校对规则，基于本书的定位及篇幅，这些内容在本书中没有体现，但是希望大家明白，即便是“字符集”这么一个看起来如此简单的知识点，在 MySQL 的知识体系中都有多处与之关联，还有很多的内容，需要大家花时间去揣摩和尝试。我们才刚刚起步，不能自满，继续努力吧！

看完本章，对字符集的设定应该有个大致了解了，那么接下来你是否在想，现在可以去数据库中创建或操作对象了吧！当然可以，其实创建对象随时都可以，前面讲字符集时不是也创建过对象嘛。不过我想很多朋友应当也听说过，MySQL 数据库中极具特色的插件式存储引擎的设置，怎么前面的创建表对象示例中没有体现呢？这个嘛，我想先多问一句，知道创建表对象时要为它指定什么存储引擎不？怎么，你不知道都有哪些存储引擎，嘿嘿，没关系，继续往下翻吧，咱们马上就要讲到这个主题。