# 第15章_存储过程与函数

MySQL从5.0版本开始支持存储过程和函数。存储过程和函数能够将复杂的SQL逻辑封装在一起，应用程序无须关注存储过程和函数内部复杂的SQL逻辑，而只需要简单地调用存储过程和函数即可。

## 1. 存储过程概述

### 1.1 理解

**含义**：存储过程的英文是 `Stored Procedure` 。它的思想很简单，就是一组经过 `预先编译` 的 SQL 语句的封装。

执行过程：存储过程预先存储在 MySQL 服务器上，需要执行的时候，客户端只需要向服务器端发出调用存储过程的命令，服务器端就可以把预先存储好的这一系列 SQL 语句全部执行。

**好处**：

1、简化操作，提高了sql语句的重用性，减少了开发程序员的压力 2、减少操作过程中的失误，提高效率 3、减少网络传输量（客户端不需要把所有的 SQL 语句通过网络发给服务器） 4、减少了 SQL 语句暴露在网上的风险，也提高了数据查询的安全性

**和视图、函数的对比**：

它和视图有着同样的优点，清晰、安全，还可以减少网络传输量。不过它和视图不同，视图是 `虚拟表` ，通常不对底层数据表直接操作，而存储过程是程序化的 SQL，可以 `直接操作底层数据表` ，相比于面向集合的操作方式，能够实现一些更复杂的数据处理。

一旦存储过程被创建出来，使用它就像使用函数一样简单，我们直接通过调用存储过程名即可。相较于函数，存储过程是 `没有返回值` 的。

### 1.2 分类

存储过程的参数类型可以是IN、OUT和INOUT。根据这点分类如下：

1、没有参数（无参数无返回） 2、仅仅带 IN 类型（有参数无返回） 3、仅仅带 OUT 类型（无参数有返回） 4、既带 IN 又带 OUT（有参数有返回） 5、带 INOUT（有参数有返回）

注意：IN、OUT、INOUT 都可以在一个存储过程中带多个。

## 2. 创建存储过程

## 2.1 语法分析

语法：

```
CREATE PROCEDURE 存储过程名(IN|OUT|INOUT 参数名 参数类型,...)
[characteristics ...]
BEGIN
    存储过程体

END
```

类似于Java中的方法：

```
修饰符 返回类型 方法名(参数类型 参数名,...){

    方法体;
}
```

说明：

1、参数前面的符号的意思

- `IN`：当前参数为输入参数，也就是表示入参；

  存储过程只是读取这个参数的值。如果没有定义参数种类，`默认就是 IN`，表示输入参数。
- `OUT`：当前参数为输出参数，也就是表示出参；

  执行完成之后，调用这个存储过程的客户端或者应用程序就可以读取这个参数返回的值了。
- `INOUT`：当前参数既可以为输入参数，也可以为输出参数。

2、形参类型可以是 MySQL数据库中的任意类型。

3、`characteristics` 表示创建存储过程时指定的对存储过程的约束条件，其取值信息如下：

```
LANGUAGE SQL
| [NOT] DETERMINISTIC
| { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
| SQL SECURITY { DEFINER | INVOKER }
| COMMENT 'string'
```

- `LANGUAGE SQL`：说明存储过程执行体是由SQL语句组成的，当前系统支持的语言为SQL。
- `[NOT] DETERMINISTIC`：指明存储过程执行的结果是否确定。DETERMINISTIC表示结果是确定的。每次执行存储过程时，相同的输入会得到相同的输出。NOT DETERMINISTIC表示结果是不确定的，相同的输入可能得到不同的输出。如果没有指定任意一个值，默认为NOT DETERMINISTIC。
- `{ CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }`：指明子程序使用SQL语句的限制。
  - CONTAINS SQL表示当前存储过程的子程序包含SQL语句，但是并不包含读写数据的SQL语句；
  - NO SQL表示当前存储过程的子程序中不包含任何SQL语句；
  - READS SQL DATA表示当前存储过程的子程序中包含读数据的SQL语句；
  - MODIFIES SQL DATA表示当前存储过程的子程序中包含写数据的SQL语句。
  - 默认情况下，系统会指定为CONTAINS SQL。
- `SQL SECURITY { DEFINER | INVOKER }`：执行当前存储过程的权限，即指明哪些用户能够执行当前存储过程。
  - `DEFINER` 表示只有当前存储过程的创建者或者定义者才能执行当前存储过程；
  - `INVOKER` 表示拥有当前存储过程的访问权限的用户能够执行当前存储过程。
  - 如果没有设置相关的值，则MySQL默认指定值为DEFINER。

- `COMMENT 'string'`：注释信息，可以用来描述存储过程。

4、存储过程体中可以有多条 SQL 语句，如果仅仅一条SQL 语句，则可以省略 BEGIN 和 END

编写存储过程并不是一件简单的事情，可能存储过程中需要复杂的 SQL 语句。

```
1. BEGIN…END：BEGIN…END 中间包含了多个语句，每个语句都以（;）号为结束符。
2. DECLARE：DECLARE 用来声明变量，使用的位置在于 BEGIN…END 语句中间，而且需要在其他语句使用之前进行变量的声明。
3. SET：赋值语句，用于对变量进行赋值。
4. SELECT… INTO：把从数据表中查询的结果存放到变量中，也就是为变量赋值。
```

5、需要设置新的结束标记

```
DELIMITER 新的结束标记
```

因为MySQL默认的语句结束符号为分号‘;’。为了避免与存储过程中SQL语句结束符相冲突，需要使用DELIMITER改变存储过程的结束符。

比如：“DELIMITER //”语句的作用是将MySQL的结束符设置为//，并以“END //”结束存储过程。存储过程定义完毕之后再使用“DELIMITER ;”恢复默认结束符。DELIMITER也可以指定其他符号作为结束符。

当使用DELIMITER命令时，应该避免使用反斜杠（‘\’）字符，因为反斜线是MySQL的转义字符。

示例：

```
DELIMITER $

CREATE PROCEDURE 存储过程名(IN|OUT|INOUT 参数名  参数类型,...)
[characteristics ...]
BEGIN
    sql语句1;
    sql语句2;

END $
```

## 2.2 代码举例

举例1：创建存储过程select_all_data()，查看 emps 表的所有数据

```
DELIMITER $

CREATE PROCEDURE select_all_data()
BEGIN
    SELECT * FROM emps;

END $

DELIMITER ;
```

举例2：创建存储过程avg_employee_salary()，返回所有员工的平均工资

```
DELIMITER //

CREATE PROCEDURE avg_employee_salary ()
BEGIN
    SELECT AVG(salary) AS avg_salary FROM emps;
END //

DELIMITER ;
```

举例3：创建存储过程show_max_salary()，用来查看“emps”表的最高薪资值。

```
CREATE PROCEDURE show_max_salary()
    LANGUAGE SQL
    NOT DETERMINISTIC
    CONTAINS SQL
    SQL SECURITY DEFINER
    COMMENT '查看最高薪资'
    BEGIN
        SELECT MAX(salary) FROM emps;
    END //

DELIMITER ;
```

举例4：创建存储过程show_min_salary()，查看“emps”表的最低薪资值。并将最低薪资通过OUT参数“ms”输出

```
DELIMITER //

CREATE PROCEDURE show_min_salary(OUT ms DOUBLE)
    BEGIN
        SELECT MIN(salary) INTO ms FROM emps;
    END //

DELIMITER ;
```

举例5：创建存储过程show_someone_salary()，查看“emps”表的某个员工的薪资，并用IN参数empname输入员工姓名。

```
DELIMITER //

CREATE PROCEDURE show_someone_salary(IN empname VARCHAR(20))
    BEGIN
        SELECT salary FROM emps WHERE ename = empname;
    END //

DELIMITER ;
```

举例6：创建存储过程show_someone_salary2()，查看“emps”表的某个员工的薪资，并用IN参数empname输入员工姓名，用OUT参数empsalary输出员工薪资。

```
DELIMITER //

CREATE PROCEDURE show_someone_salary2(IN empname VARCHAR(20),OUT empsalary DOUBLE)
    BEGIN
        SELECT salary INTO empsalary FROM emps WHERE ename = empname;
    END //

DELIMITER ;
```

举例7：创建存储过程show_mgr_name()，查询某个员工领导的姓名，并用INOUT参数“empname”输入员工姓名，输出领导的姓名。

```
DELIMITER //

CREATE PROCEDURE show_mgr_name(INOUT empname VARCHAR(20))
    BEGIN
        SELECT ename INTO empname FROM emps
        WHERE eid = (SELECT MID FROM emps WHERE ename=empname);
    END //

DELIMITER ;
```

# 3. 调用存储过程

## 3.1 调用格式

存储过程有多种调用方法。存储过程必须使用CALL语句调用，并且存储过程和数据库相关，如果要执行其他数据库中的存储过程，需要指定数据库名称，例如CALL dbname.procname。

```
CALL 存储过程名(实参列表)
```

**格式：**

1、调用in模式的参数：

```
CALL sp1('值');
```

2、调用out模式的参数：

```
SET @name;
CALL sp1(@name);
SELECT @name;
```

3、调用inout模式的参数：

```
SET @name=值;
CALL sp1(@name);
SELECT @name;
```

## 3.2 代码举例

**举例1：**

```
DELIMITER //

CREATE PROCEDURE CountProc(IN sid INT,OUT num INT)
BEGIN
    SELECT COUNT(*) INTO num FROM fruits
    WHERE s_id = sid;
END //

DELIMITER ;
```

调用存储过程：

```
mysql> CALL CountProc (101, @num);
Query OK, 1 row affected (0.00 sec)
```

查看返回结果：

```
mysql> SELECT @num;
```

该存储过程返回了指定 s_id=101 的水果商提供的水果种类，返回值存储在num变量中，使用SELECT查看，返回结果为3。

**举例2：** 创建存储过程，实现累加运算，计算 1+2+…+n 等于多少。具体的代码如下：

```
DELIMITER //
CREATE PROCEDURE `add_num`(IN n INT)
BEGIN
       DECLARE i INT;
       DECLARE sum INT;

       SET i = 1;
       SET sum = 0;
       WHILE i <= n DO
              SET sum = sum + i;
              SET i = i +1;
       END WHILE;
       SELECT sum;
END //
DELIMITER ;
```

如果你用的是 Navicat 工具，那么在编写存储过程的时候，Navicat 会自动设置 DELIMITER 为其他符号，我们不需要再进行 DELIMITER 的操作。

直接使用 `CALL add_num(50);` 即可。这里我传入的参数为 50，也就是统计 1+2+…+50 的积累之和。

## 3.3 如何调试

在 MySQL 中，存储过程不像普通的编程语言（比如 VC++、Java 等）那样有专门的集成开发环境。因此，你可以通过 SELECT 语句，把程序执行的中间结果查询出来，来调试一个 SQL 语句的正确性。调试成功之后，把 SELECT 语句后移到下一个 SQL 语句之后，再调试下一个 SQL 语句。这样 逐步推进 ，就可以完成对存储过程中所有操作的调试了。当然，你也可以把存储过程中的 SQL 语句复制出来，逐段单独调试。

# 4. 存储函数的使用

前面学习了很多函数，使用这些函数可以对数据进行的各种处理操作，极大地提高用户对数据库的管理效率。MySQL支持自定义函数，定义好之后，调用方式与调用MySQL预定义的系统函数一样。

## 4.1 语法分析

学过的函数：LENGTH、SUBSTR、CONCAT等

语法格式：

```
CREATE FUNCTION 函数名(参数名 参数类型,...)
RETURNS 返回值类型
[characteristics ...]
BEGIN
    函数体    #函数体中肯定有 RETURN 语句

END
```

说明：

1、参数列表：指定参数为IN、OUT或INOUT只对PROCEDURE是合法的，FUNCTION中总是默认为IN参数。

2、RETURNS type 语句表示函数返回数据的类型；

RETURNS子句只能对FUNCTION做指定，对函数而言这是`强制`的。它用来指定函数的返回类型，而且函数体必须包含一个`RETURN value`语句。

3、characteristic 创建函数时指定的对函数的约束。取值与创建存储过程时相同，这里不再赘述。

4、函数体也可以用BEGIN...END来表示SQL代码的开始和结束。如果函数体只有一条语句，也可以省略BEGIN...END。

## 4.2 调用存储函数

在MySQL中，存储函数的使用方法与MySQL内部函数的使用方法是一样的。换言之，用户自己定义的存储函数与MySQL内部函数是一个性质的。区别在于，存储函数是`用户自己定义`的，而内部函数是MySQL的`开发者定义`的。

```
SELECT 函数名(实参列表)
```

## 4.3 代码举例

**举例1：**

创建存储函数，名称为email_by_name()，参数定义为空，该函数查询Abel的email，并返回，数据类型为字符串型。

```
DELIMITER //

CREATE FUNCTION email_by_name()
RETURNS VARCHAR(25)
DETERMINISTIC
CONTAINS SQL
BEGIN
    RETURN (SELECT email FROM employees WHERE last_name = 'Abel');
END //

DELIMITER ;
```

调用：

```
SELECT email_by_name();
```

**举例2：**

创建存储函数，名称为email_by_id()，参数传入emp_id，该函数查询emp_id的email，并返回，数据类型为字符串型。

```
DELIMITER //

CREATE FUNCTION email_by_id(emp_id INT)
RETURNS VARCHAR(25)
DETERMINISTIC
CONTAINS SQL
BEGIN
    RETURN (SELECT email FROM employees WHERE employee_id = emp_id);
END //

DELIMITER ;
```

调用：

```
SET @emp_id = 102;
SELECT email_by_id(102);
```

**举例3：**

创建存储函数count_by_id()，参数传入dept_id，该函数查询dept_id部门的员工人数，并返回，数据类型为整型。

```
DELIMITER //

CREATE FUNCTION count_by_id(dept_id INT)
RETURNS INT
    LANGUAGE SQL
    NOT DETERMINISTIC
    READS SQL DATA
    SQL SECURITY DEFINER
    COMMENT '查询部门平均工资'
BEGIN
    RETURN (SELECT COUNT(*) FROM employees WHERE department_id = dept_id);

END //

DELIMITER ;
```

调用：

```
SET @dept_id = 50;
SELECT count_by_id(@dept_id);
```

**注意：**

若在创建存储函数中报错“`you might want to use the less safe log_bin_trust_function_creators variable`”，有两种处理方法：

- 方式1：加上必要的函数特性“[NOT] DETERMINISTIC”和“{CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA}”
- 方式2：

```
mysql> SET GLOBAL log_bin_trust_function_creators = 1;
```

### 4.4 对比存储函数和存储过程

| | 关键字 | 调用语法 | 返回值 | 应用场景 |
| --- | --- | --- | --- | --- |
| 存储过程 | PROCEDURE | CALL 存储过程() | 理解为有0个或多个 | 一般用于更新 |
| 存储函数 | FUNCTION | SELECT 函数() | 只能是一个 | 一般用于查询结果为一个值并返回时 |

此外，**存储函数可以放在查询语句中使用，存储过程不行**。反之，存储过程的功能更加强大，包括能够执行对表的操作（比如创建表，删除表等）和事务操作，这些功能是存储函数不具备的。

## 5. 存储过程和函数的查看、修改、删除

### 5.1 查看

创建完之后，怎么知道我们创建的存储过程、存储函数是否成功了呢？

MySQL存储了存储过程和函数的状态信息，用户可以使用SHOW STATUS语句或SHOW CREATE语句来查看，也可直接从系统的information_schema数据库中查询。这里介绍3种方法。

**1. 使用SHOW CREATE语句查看存储过程和函数的创建信息**

基本语法结构如下：

```
SHOW CREATE {PROCEDURE | FUNCTION} 存储过程名或函数名
```

举例：

```
SHOW CREATE FUNCTION test_db.CountProc \G
```

**2. 使用SHOW STATUS语句查看存储过程和函数的状态信息**

基本语法结构如下：

```
SHOW {PROCEDURE | FUNCTION} STATUS [LIKE 'pattern']
```

这个语句返回子程序的特征，如数据库、名字、类型、创建者及创建和修改日期。

[LIKE 'pattern']：匹配存储过程或函数的名称，可以省略。当省略不写时，会列出MySQL数据库中存在的所有存储过程或函数的信息。 举例：SHOW STATUS语句示例，代码如下：

```
mysql> SHOW PROCEDURE STATUS LIKE 'SELECT%' \G
*************************** 1. row ***************************
                  Db: test_db
                Name: SelectAllData
                Type: PROCEDURE
             Definer: root@localhost
            Modified: 2021-10-16 15:55:07
             Created: 2021-10-16 15:55:07
       Security_type: DEFINER
             Comment:
character_set_client: utf8mb4
collation_connection: utf8mb4_general_ci
  Database Collation: utf8mb4_general_ci
1 row in set (0.00 sec)
```

#### 3. 从information_schema.Routines表中查看存储过程和函数的信息

MySQL中存储过程和函数的信息存储在information_schema数据库下的Routines表中。可以通过查询该表的记录来查询存储过程和函数的信息。其基本语法形式如下：

```
SELECT * FROM information_schema.Routines
WHERE ROUTINE_NAME='存储过程或函数的名' [AND ROUTINE_TYPE = {'PROCEDURE|FUNCTION'}];
```

说明：如果在MySQL数据库中存在存储过程和函数名称相同的情况，最好指定ROUTINE_TYPE查询条件来指明查询的是存储过程还是函数。

举例：从Routines表中查询名称为CountProc的存储函数的信息，代码如下：

```
SELECT * FROM information_schema.Routines
WHERE ROUTINE_NAME='count_by_id'  AND  ROUTINE_TYPE = 'FUNCTION' \G
```

## 5.2 修改

修改存储过程或函数，不影响存储过程或函数功能，只是修改相关特性。使用ALTER语句实现。

```
ALTER {PROCEDURE | FUNCTION} 存储过程或函数的名 [characteristic ...]
```

其中，characteristic指定存储过程或函数的特性，其取值信息与创建存储过程、函数时的取值信息略有不同。

```
{ CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
| SQL SECURITY { DEFINER | INVOKER }
| COMMENT 'string'
```

- `CONTAINS SQL`，表示子程序包含SQL语句，但不包含读或写数据的语句。
- `NO SQL`，表示子程序中不包含SQL语句。
- `READS SQL DATA`，表示子程序中包含读数据的语句。
- `MODIFIES SQL DATA`，表示子程序中包含写数据的语句。
- `SQL SECURITY { DEFINER | INVOKER }`，指明谁有权限来执行。
  - `DEFINER`，表示只有定义者自己才能够执行。
  - `INVOKER`，表示调用者可以执行。
- `COMMENT 'string'`，表示注释信息。

> 修改存储过程使用ALTER PROCEDURE语句，修改存储函数使用ALTER FUNCTION语句。但是，这两个语句的结构是一样的，语句中的所有参数也是一样的。

**举例1：**

修改存储过程CountProc的定义。将读写权限改为MODIFIES SQL DATA，并指明调用者可以执行，代码如下：

```
ALTER  PROCEDURE  CountProc
MODIFIES SQL DATA
SQL SECURITY INVOKER ;
```

查询修改后的信息：

```
SELECT specific_name,sql_data_access,security_type
FROM information_schema.`ROUTINES`
WHERE routine_name = 'CountProc' AND routine_type = 'PROCEDURE';
```

结果显示，存储过程修改成功。从查询的结果可以看出，访问数据的权限（SQL_DATA_ ACCESS）已经变成MODIFIES SQL DATA，安全类型（SECURITY_TYPE）已经变成INVOKER。

**举例2：**

修改存储函数CountProc的定义。将读写权限改为READS SQL DATA，并加上注释信息“FIND NAME”，代码如下：

```
ALTER  FUNCTION  CountProc
READS SQL DATA
COMMENT 'FIND NAME' ;
```

存储函数修改成功。从查询的结果可以看出，访问数据的权限（SQL_DATA_ACCESS）已经变成READS SQL DATA，函数注释（ROUTINE_COMMENT）已经变成FIND NAME。

### 5.3 删除

删除存储过程和函数，可以使用DROP语句，其语法结构如下：

```
DROP {PROCEDURE | FUNCTION} [IF EXISTS] 存储过程或函数的名
```

IF EXISTS：如果程序或函数不存储，它可以防止发生错误，产生一个用SHOW WARNINGS查看的警告。

举例：

```
DROP PROCEDURE CountProc;
```

```
DROP FUNCTION CountProc;
```

## 6. 关于存储过程使用的争议

尽管存储过程有诸多优点，但是对于存储过程的使用，**一直都存在着很多争议**，比如有些公司对于大型项目要求使用存储过程，而有些公司在手册中明确禁止使用存储过程，为什么这些公司对存储过程的使用需求差别这么大呢？

## 6.1 优点

**1、存储过程可以一次编译多次使用。**存储过程只在创建时进行编译，之后的使用都不需要重新编译，这就提升了 SQL 的执行效率。

**2、可以减少开发工作量。**将代码 `封装` 成模块，实际上是编程的核心思想之一，这样可以把复杂的问题拆解成不同的模块，然后模块之间可以 `重复使用` ，在减少开发工作量的同时，还能保证代码的结构清晰。

**3、存储过程的安全性强。**我们在设定存储过程的时候可以 `设置对用户的使用权限` ，这样就和视图一样具有较强的安全性。

**4、可以减少网络传输量。**因为代码封装到存储过程中，每次使用只需要调用存储过程即可，这样就减少了网络传输量。

**5、良好的封装性。**在进行相对复杂的数据库操作时，原本需要使用一条一条的 SQL 语句，可能要连接多次数据库才能完成的操作，现在变成了一次存储过程，只需要 `连接一次即可` 。

## 6.2 缺点

基于上面这些优点，不少大公司都要求大型项目使用存储过程，比如微软、IBM 等公司。但是国内的阿里并不推荐开发人员使用存储过程，这是为什么呢?

> **阿里开发规范**
>
> 【强制】禁止使用存储过程，存储过程难以调试和扩展，更没有移植性。

存储过程虽然有诸如上面的好处，但缺点也是很明显的。

**1、可移植性差。**存储过程不能跨数据库移植，比如在 MySQL、Oracle 和 SQL Server 里编写的存储过程，在换成其他数据库时都需要重新编写。

**2、调试困难。**只有少数 DBMS 支持存储过程的调试。对于复杂的存储过程来说，开发和维护都不容易。虽然也有一些第三方工具可以对存储过程进行调试，但要收费。

**3、存储过程的版本管理很困难。**比如数据表索引发生变化了，可能会导致存储过程失效。我们在开发软件的时候往往需要进行版本管理，但是存储过程本身没有版本控制，版本迭代更新的时候很麻烦。

**4、它不适合高并发的场景。**高并发的场景需要减少数据库的压力，有时数据库会采用分库分表的方式，而且对可扩展性要求很高，在这种情况下，存储过程会变得难以维护，`增加数据库的压力` ，显然就不适用了。

小结:

存储过程既方便，又有局限性。尽管不同的公司对存储过程的态度不一，但是对于我们开发人员来说，不论怎样，掌握存储过程都是必备的技能之一。