13丨什么是存储过程，在实际项目中用得多么？

上一节我介绍了视图，它是 SQL 中的一个重要应用，使用视图对 SQL 查询进行封装，可以让 SQL 的代码结构更清晰，让用户权限管理更安全。

今天我来讲一下 SQL 的存储过程，它是 SQL 中另一个重要应用，和视图一样，都是对 SQL 代码进行封装，可以反复利用。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。存储过程可以说是由 SQL 语句和流控制语句构成的语句集合，它和我们之前学到的函数一样，可以接收输入参数，也可以返回输出参数给调用者，返回计算结果。

今天有关存储过程的内容，你将重点掌握以下几个部分：

1. 什么是存储过程，如何创建一个存储过程？
2. 流控制语句都有哪些，如何使用它们？
3. 各大公司是如何看待存储过程的？在实际工作中，我们该如何使用存储过程？

什么是存储过程，如何创建一个存储过程

存储过程的英文是 Stored Procedure。它的思想很简单，就是 SQL 语句的封装。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。我在前面讲过，存储过程实际上由 SQL 语句和流控制语句共同组成。流控制语句都有哪些呢？这个我稍后讲解。

我们先来看下如何定义一个存储过程：

复制代码

```
CREATE PROCEDURE 存储过程名称 ([参数列表])
BEGIN
    需要执行的语句
END
```

在这里，我们使用 CREATE PROCEDURE 创建一个存储过程，后面是存储过程的名称，以及过程所带的参数，可以包括输入参数和输出参数。最后由 BEGIN 和 END 来定义我们所

要执行的语句块。

和视图一样，我们可以删除已经创建的存储过程，使用的是 DROP PROCEDURE。如果要更新存储过程，我们需要使用 ALTER PROCEDURE。

讲完了如何创建，更新和删除一个存储过程，下面我们来看下如何实现一个简单的存储过程。比如我想做一个累加运算，计算 1+2+...+n 等于多少，我们可以通过参数 n 来表示想要累加的个数，那么如何用存储过程实现这一目的呢？这里我做一个 add_num 的存储过程，具体的代码如下：

复制代码

```
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
```

当我们需要再次使用这个存储过程的时候，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，查询结果为：

sum
1275

这就是一个简单的存储过程，除了理解 1+2+...+n 的实现过程，还有两点你需要理解，一个是 DELIMITER 定义语句的结束符，另一个是存储过程的三种参数类型。

我们先来看下 DELIMITER 的作用。如果你使用 Navicat 这个工具来管理 MySQL 执行存储过程，那么直接执行上面这段代码就可以了。如果用的是 MySQL，你还需要用

DELIMITER 来临时定义新的结束符。因为默认情况下 SQL 采用（；）作为结束符，这样当存储过程中的每一句 SQL 结束之后，采用（；）作为结束符，就相当于告诉 SQL 可以执行这一句了。但是存储过程是一个整体，我们不希望 SQL 逐条执行，而是采用存储过程整段执行的方式，因此我们就需要临时定义新的 DELIMITER，新的结束符可以用（//）或者（$$）。如果你用的是 MySQL，那么上面这段代码，应该写成下面这样：

复制代码

```
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 ;
```

首先我用（//）作为结束符，又在整个存储过程结束后采用了（//）作为结束符号，告诉 SQL 可以执行了，然后再将结束符还原成默认的（;）。

需要注意的是，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。

我们再来看下存储过程的 3 种参数类型。在刚才的存储过程中，我们使用了 IN 类型的参数，另外还有 OUT 类型和 INOUT 类型，作用如下：

参数类型	是否返回	作用
IN	否	向存储过程传入参数，存储过程中修改该参数的值，不能被返回。
OUT	是	把存储过程计算的结果放到该参数中，调用者可以得到返回值。
INOUT	是	IN和OUT的结合，既用于存储过程的传入参数，同时又可以把计算结果放到参数中，调用者可以得到返回值。

IN 和 OUT 的结合，既用于存储过程的传入参数，同时又可以把计算结果放到参数中，调用者可以得到返回值。

你能看到，IN 参数必须在调用存储过程时指定，而在存储过程中修改该参数的值不能被返回。而 OUT 参数和 INOUT 参数可以在存储过程中被改变，并可返回。

举个例子，这里会用到我们之前讲过的王者荣耀的英雄数据表 heros。假设我想创建一个存储类型 get_hero_scores，用来查询某一类型英雄中的最大的最大生命值，最小的最大魔法值，以及平均最大攻击值，那么该怎么写呢？

复制代码

```
CREATE PROCEDURE `get_hero_scores`(
        OUT max_max_hp FLOAT,
        OUT min_max_mp FLOAT,
        OUT avg_max_attack FLOAT,
        s VARCHAR(255)
        )
BEGIN
        SELECT MAX(hp_max), MIN(mp_max), AVG(attack_max) FROM heros WHERE role_main = s 
END
```

你能看到我定义了 4 个参数类型，其中 3 个为 OUT 类型，分别为 max_max_hp、min_max_mp 和 avg_max_attack，另一个参数 s 为 IN 类型。

这里我们从 heros 数据表中筛选主要英雄定位为 s 的英雄数据，即筛选条件为 role_main=s，提取这些数据中的最大的最大生命值，最小的最大魔法值，以及平均最大攻击值，分别赋值给变量 max_max_hp、min_max_mp 和 avg_max_attack。

然后我们就可以调用存储过程，使用下面这段代码即可：

复制代码

```
CALL get_hero_scores(@max_max_hp, @min_max_mp, @avg_max_attack, '战士');
SELECT @max_max_hp, @min_max_mp, @avg_max_attack;
```

运行结果：

@max_max_hp	@min_max_mp	@avg_max_attack
8050	0	342.1666564941406

流控制语句

流控制语句是用来做流程控制的，我刚才讲了两个简单的存储过程的例子，一个是 1+2+...+n 的结果计算，一个是王者荣耀的数据查询，你能看到这两个例子中，我用到了下面的流控制语句：

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。

除了上面这些用到的流控制语句以外，还有一些常用的流控制语句：

1.IF...THEN...ENDIF：条件判断语句，我们还可以在 IF...THEN...ENDIF 中使用 ELSE 和 ELSEIF 来进行条件判断。
2.CASE：CASE 语句用于多条件的分支判断，使用的语法是下面这样的。

复制代码

```
CASE
        WHEN expression1 THEN ...
        WHEN expression2 THEN ...
        ...
    ELSE
```

```
    --ELSE 语句可以加，也可以不加。加的话代表的所有条件都不满足时采用的方式。
END
```

3.LOOP、LEAVE 和 ITERATE：LOOP 是循环语句，使用 LEAVE 可以跳出循环，使用 ITERATE 则可以进入下一次循环。如果你有面向过程的编程语言的使用经验，你可以把 LEAVE 理解为 BREAK，把 ITERATE 理解为 CONTINUE。
4.REPEAT...UNTIL...END REPEAT：这是一个循环语句，首先会执行一次循环，然后在 UNTIL 中进行表达式的判断，如果满足条件就退出，即 END REPEAT；如果条件不满足，则会就继续执行循环，直到满足退出条件为止。
5.WHILE...DO...END WHILE：这也是循环语句，和 REPEAT 循环不同的是，这个语句需要先进行条件判断，如果满足条件就进行循环，如果不满足条件就退出循环。

我们之前说过 SQL 是声明型语言，使用 SQL 就像在使用英语，简单直接。今天讲的存储过程，尤其是在存储过程中使用到的流控制语句，属于过程性语言，类似于 C++ 语言中函数，这些语句可以帮我们解决复杂的业务逻辑。

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？

我们得从存储过程的特点来找答案。

你能看到存储过程有很多好处。

首先存储过程可以一次编译多次使用。存储过程只在创造时进行编译，之后的使用都不需要重新编译，这就提升了 SQL 的执行效率。其次它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。还有一点，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。最后它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。同时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。

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？

存储过程虽然有诸如上面的好处，但缺点也是很明显的。

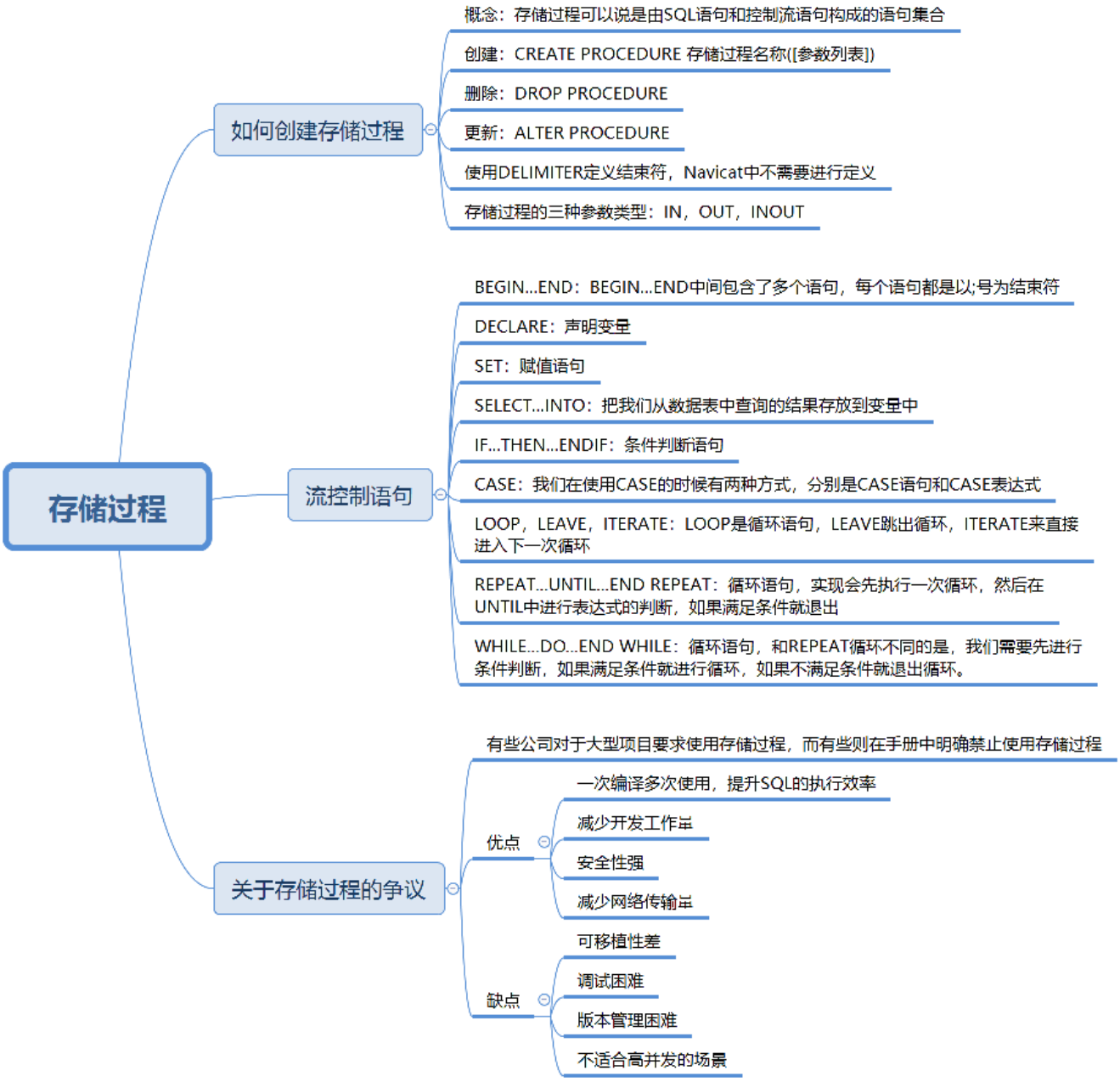
它的可移植性差，存储过程不能跨数据库移植，比如在 MySQL、Oracle 和 SQL Server 里编写的存储过程，在换成其他数据库时都需要重新编写。

其次调试困难，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。

此外，存储过程的版本管理也很困难，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。

最后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。

了解了存储过程的优缺点之后，我想说的是，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。



最后我们做一个小练习吧。针对王者荣耀的英雄数据表 heros 表，请编写存储过程 get_sum_score，用来得到某一类型英雄（主要定位为某一类型即可）的最大生命值的总和。