查询操作主要是PDO::query()、PDO::exec()、PDO::prepare()。PDO::query()主要是用于有记录结果返回的操作，特别是SELECT操作，PDO::exec()主要是针对没有结果集合返回的操作，比如INSERT、UPDATE、DELETE等操作，它返回的结果是当前操作影响的列数。PDO::prepare()主要是预处理操作，需要通过$rs->execute()来执行预处理里面的SQL语句，这个方法可以绑定参数，功能比较强大，不是本文能够简单说明白的，大家可以参考手册和其他文档。

获取结果集操作主要是：PDOStatement::fetchColumn()、PDOStatement::fetch()、PDOStatement::fetchALL()。PDOStatement::fetchColumn() 是获取结果指定第一条记录的某个字段，缺省是第一个字段。PDOStatement::fetch() 是用来获取一条记录，PDOStatement::fetchAll()是获取所有记录集到一个中，获取结果可以通过PDOStatement::setFetchMode来设置需要结果集合的类型。

另外有两个周边的操作，一个是PDO::lastInsertId()和PDOStatement::rowCount()。PDO::lastInsertId()是返回上次插入操作，主键列类型是自增的最后的自增ID。PDOStatement::rowCount()主要是用于PDO::query()和PDO::prepare()进行DELETE、INSERT、UPDATE操作影响的结果集，对PDO::exec()方法和SELECT操作无效。

**PHP PDO 预处理语句与存储过程**

[[PHP PDO 参考手册](http://www.runoob.com/php/php-pdo.html)PHP PDO 参考手册](http://www.runoob.com/php/php-pdo.html)

很多更成熟的数据库都支持预处理语句的概念。

什么是预处理语句？可以把它看作是想要运行的 SQL 的一种编译过的模板，它可以使用变量参数进行定制。预处理语句可以带来两大好处：

* 查询仅需解析（或预处理）一次，但可以用相同或不同的参数执行多次。当查询准备好后，数据库将分析、编译和优化执行该查询的计划。对于复杂的查询，此过程要花费较长的时间，如果需要以不同参数多次重复相同的查询，那么该过程将大大降低应用程序的速度。通过使用预处理语句，可以避免重复分析/编译/优化周期。简言之，预处理语句占用更少的资源，因而运行得更快。
* 提供给预处理语句的参数不需要用引号括起来，驱动程序会自动处理。如果应用程序只使用预处理语句，可以确保不会发生SQL 注入。（然而，如果查询的其他部分是由未转义的输入来构建的，则仍存在 SQL 注入的风险）。

预处理语句如此有用，以至于它们唯一的特性是在驱动程序不支持的时PDO 将模拟处理。这样可以确保不管数据库是否具有这样的功能，都可以确保应用程序可以用相同的数据访问模式。

**用预处理语句进行重复插入**

下面例子通过用 name 和 value 替代相应的命名占位符来执行一个插入查询

<?php

$stmt = $dbh->prepare("INSERT INTO REGISTRY (name, value) VALUES (:name, :value)");

$stmt->bindParam(':name', $name);

$stmt->bindParam(':value', $value);

// 插入一行

$name = 'one';

$value = 1;

$stmt->execute();

// 用不同的值插入另一行

$name = 'two';

$value = 2;

$stmt->execute();

?>

**用预处理语句进行重复插入**

下面例子通过用 name 和 value 取代 ? 占位符的位置来执行一条插入查询。

<?php

$stmt = $dbh->prepare("INSERT INTO REGISTRY (name, value) VALUES (?, ?)");

$stmt->bindParam(1, $name);

$stmt->bindParam(2, $value);

// 插入一行

$name = 'one';

$value = 1;

$stmt->execute();

// 用不同的值插入另一行

$name = 'two';

$value = 2;

$stmt->execute();

?>

**使用预处理语句获取数据**

下面例子获取数据基于键值已提供的形式。用户的输入被自动用引号括起来，因此不会有 SQL 注入攻击的危险。

<?php

$stmt = $dbh->prepare("SELECT \* FROM REGISTRY where name = ?");

if ($stmt->execute(array($\_GET['name']))) {

while ($row = $stmt->fetch()) {

print\_r($row);

}

}

?>

如果数据库驱动支持，应用程序还可以绑定输出和输入参数.输出参数通常用于从存储过程获取值。输出参数使用起来比输入参数要稍微复杂一些，因为当绑定一个输出参数时，必须知道给定参数的长度。如果为参数绑定的值大于建议的长度，就会产生一个错误。

**带输出参数调用存储过程**

<?php

$stmt = $dbh->prepare("CALL sp\_returns\_string(?)");

$stmt->bindParam(1, $return\_value, PDO::PARAM\_STR, 4000);

// 调用存储过程

$stmt->execute();

print "procedure returned $return\_value\n";

?>

还可以指定同时具有输入和输出值的参数，其语法类似于输出参数。在下一个例子中，字符串"hello"被传递给存储过程，当存储过程返回时，hello 被替换为该存储过程返回的值。

**带输入/输出参数调用存储过程**

<?php

$stmt = $dbh->prepare("CALL sp\_takes\_string\_returns\_string(?)");

$value = 'hello';

$stmt->bindParam(1, $value, PDO::PARAM\_STR|PDO::PARAM\_INPUT\_OUTPUT, 4000);

// 调用存储过程

$stmt->execute();

print "procedure returned $value\n";

?>

**占位符的无效使用**

<?php

$stmt = $dbh->prepare("SELECT \* FROM REGISTRY where name LIKE '%?%'");

$stmt->execute(array($\_GET['name']));

// 占位符必须被用在整个值的位置

$stmt = $dbh->prepare("SELECT \* FROM REGISTRY where name LIKE ?");

$stmt->execute(array("%$\_GET[name]%"));

?>