# 第1讲 PDO概述

## 概念

虽然PHP和MySQL数据库是“黄金搭档”，但是能和PHP“搭档”的不止MySQL数据库一种，还有mssql、Oracle、SQLServer，每一种数据库都为PHP提供了属于自己的操作数据库的扩展函数！比如，就以连接数据库为例：

MySQL：**mysql\_connect**

mssql：**mssql\_connect**

Oracle：**oci\_connect**

所以，如果PHP要切换不同的数据库，就要切换不同的数据库操作函数：



这样，无论从程序员学习的角度，还是从程序员开发的角度，都很麻烦！

于是，人们就在思考，能不能把对所有数据库都统一成相同的操作以实现相同的目的？

所以，PDO就问世了！

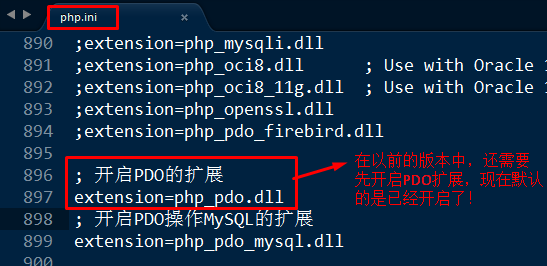
PDO的全称是：PHP Data Object（PHP数据对象）

在新版本的PHP（PHP5以上）中，PHP为用户封装了一套PDO扩展库，专门用来操作不同类型的数据库！



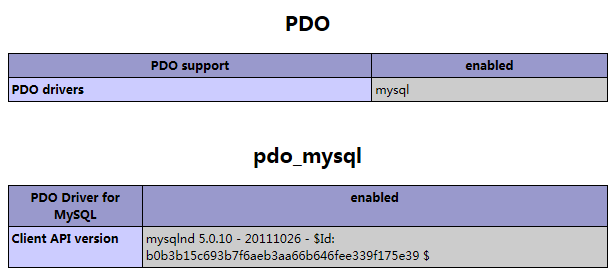
## 开启PDO扩展

同需要开启MySQL扩展和GD库扩展一样，一样需要手动的在php.ini中开启PDO扩展：



重启Apache！

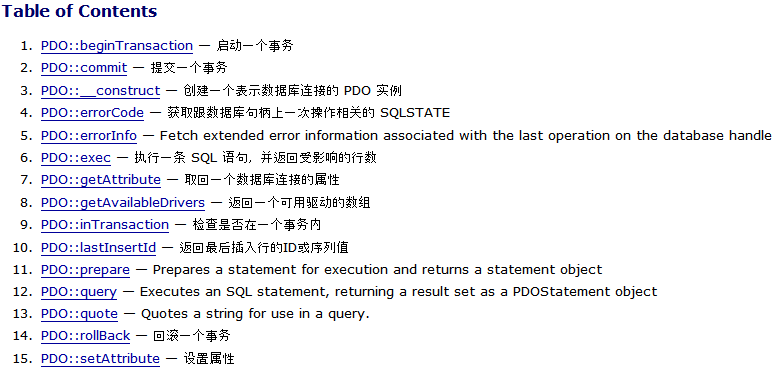
我们可以在phpinfo中看到相关的信息：



# 第2讲 PDO的实例化

由于PDO的扩展库是以面向对象的方式进行封装的，所以，我们首先要了解PDO的类结构并实例化一个对象！

## PDO类的结构



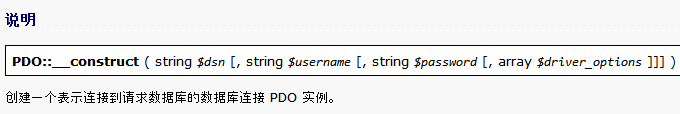
**思考：**

在PDO众多的公开方法中，我们首先应该学习的是哪一个？

**回答：**

应该是构造方法，因为构造方法就是在实例化一个对象的时候自动调用的！

## PDO构造方法



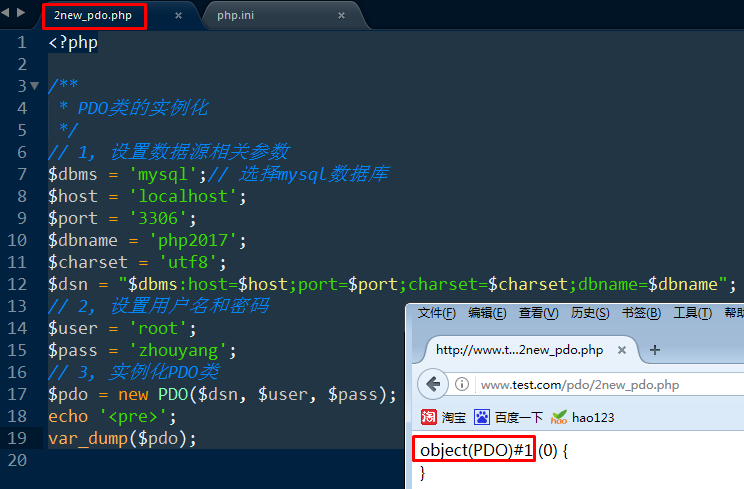
**一共有四个参数：**

$dsn：data source name数据源名称，其实就是告诉PDO类应该选择哪种类型的数据库，主机名和端口号是什么，选择哪个数据库以及采用什么样的字符编码！

$username：数据库的用户名

$password：数据库的密码

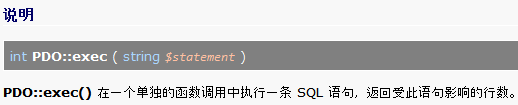
$driver\_options：附加驱动信息，暂时用默认



# 第3讲 使用PDO实现增删改操作

其实主要就是调用PDO对象的exec方法：

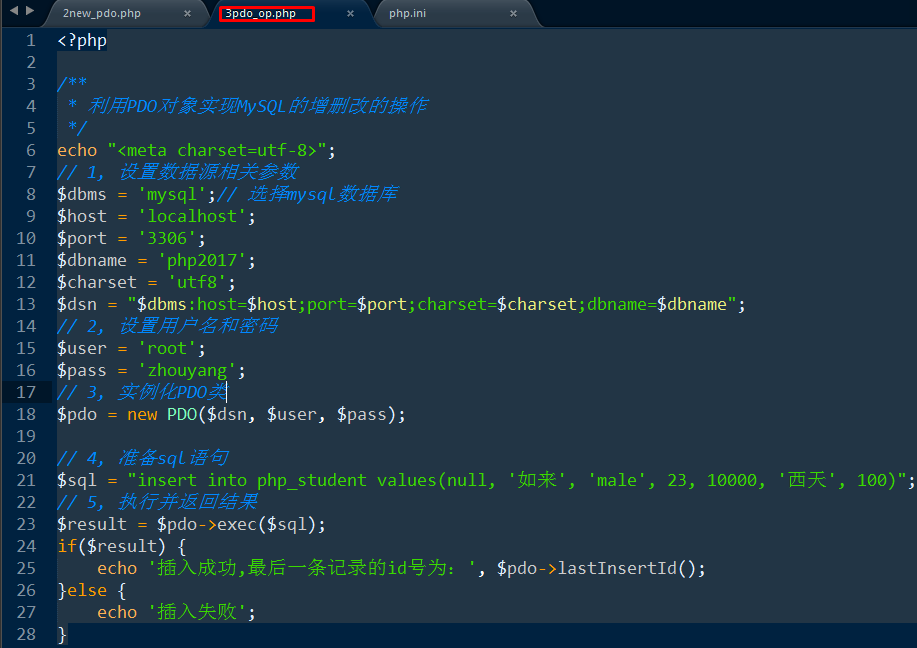
exec是execute的简写，意思为执行！



其中插入记录的时候，还有一个方法：



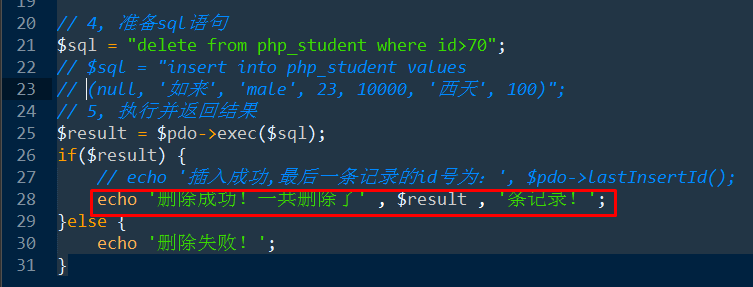
主要就是用来返回最后插入的记录的主键id号！



结果如下：



删除和更新，其实就是修改sql语句就行了！



# 第4讲 使用PDO实现查询操作

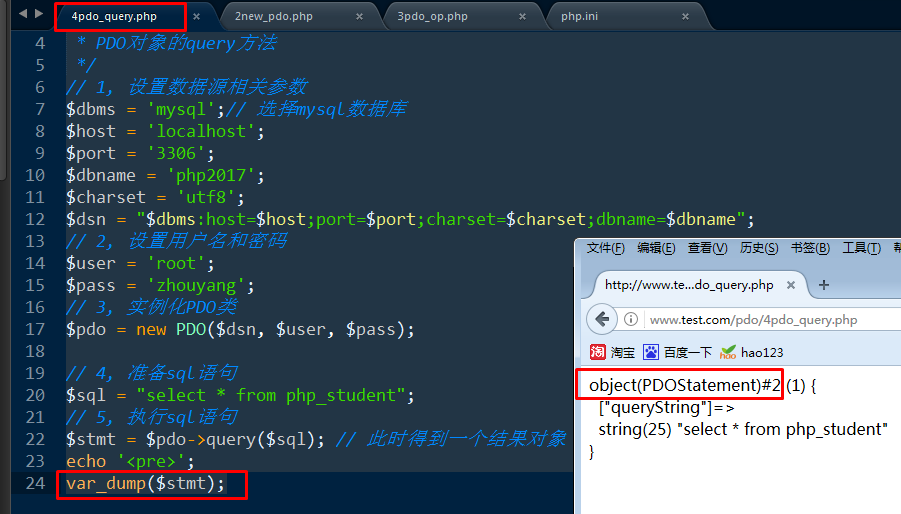
其实就是使用PDO对象中的query方法！

## query方法

PDO::query — Executes an SQL statement, returning a result set as a PDOStatement object

作用：

执行一条sql语句（一般来说，就是查询语句），并返回一个PDOStatement类的对象（类比以前的结果集资源），也叫作结果对象！





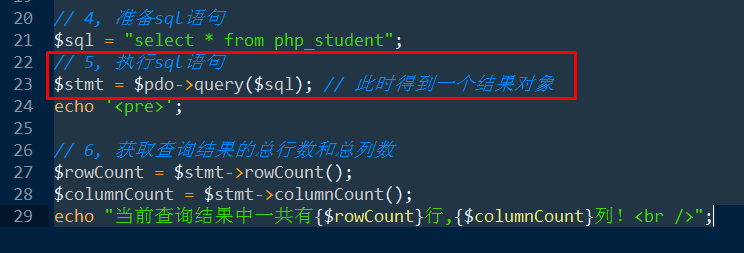
## PDOStatement对象

### rowCount

作用：获取查询结果中的总记录数（总行数）

### columnCount

作用：获取查询结果中的总字段数（总列数）



在实际的应用中，获取总记录数的操作比较多！

### fetch

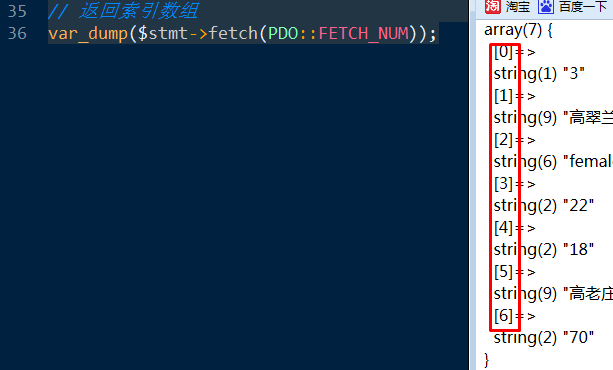
**作用：**每次从PDOStatement结果集中，获取一条记录，同时将指针下移，其中该方法有一个参数，可以控制返回值的类型！

最常用的参数如下：

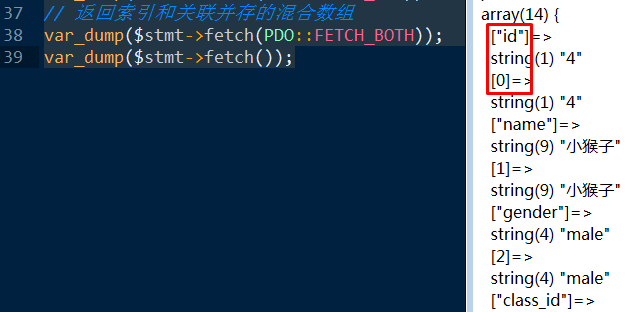
**PDO::FETCH\_ASSOC**返回一个关联型的数组，数组的下标就是字段的名字！效果类似于以前的mysql\_fetch\_assoc



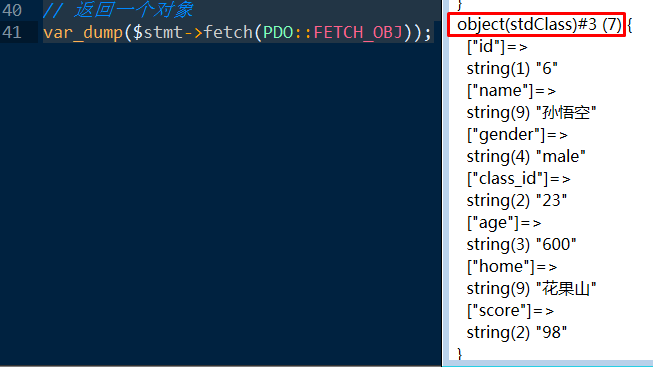
**PDO::FETCH\_NUM**返回一个索引数组，效果类似于以前的mysql\_fetch\_row



**PDO::FETCH\_BOTH**返回一个关联和索引并存的混合数组，也是缺省值！

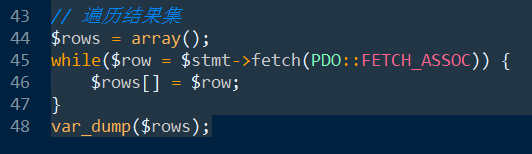


**PDO::FETCH\_OBJ**返回一个对象，其中对象的属性就是字段的值！



使用的最多其实是PDO::FETCH\_ASSOC属性！

利用fetch方法实现对结果集的遍历！

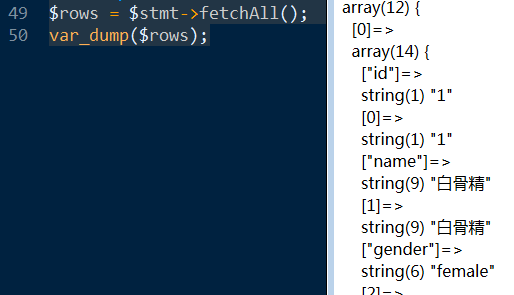


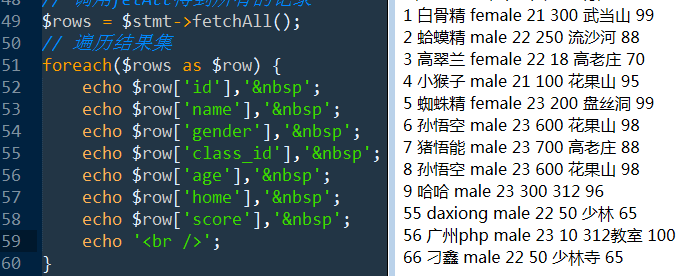
其实，有一个专门遍历结果集的方法：

### fetchAll

**作用：**

返回一个结果集中所有的记录，形成一个二维数组，比较适合多行多列的结果集的提取！



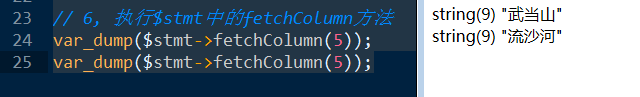


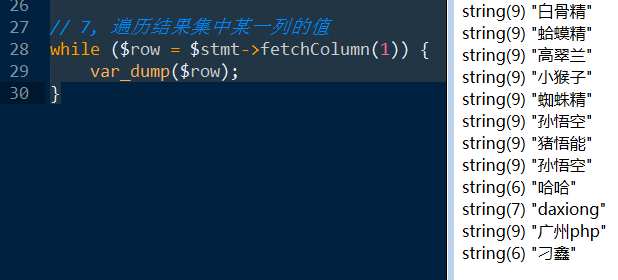
还有两个方法：

### fetchColumn

**作用：**每次获取指定的某一列信息中的一条记录，并将指针下移！

该方法需要一个参数，用来代表获取哪一列的信息，第一列参数为0，第二列为1，依次类推！

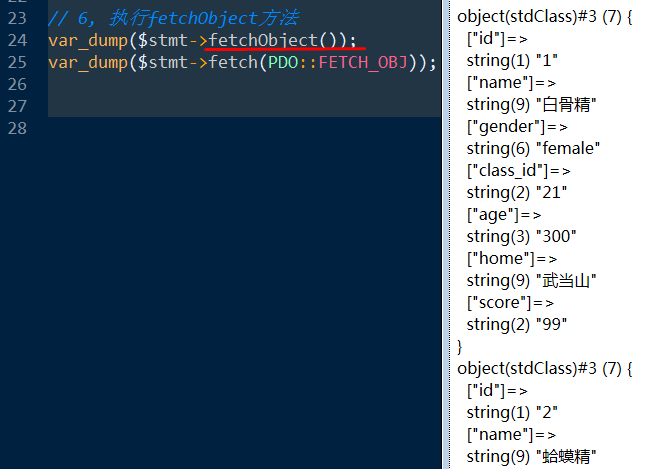




### fetchObject

**作用：**每次读取一条记录放到一个对象中，并使指针下移！

其实就是相当于fetch(PDO::FETCH\_OBJ)



# 第5讲 MySQL的预处理技术

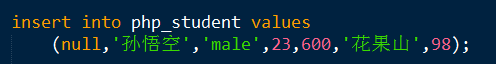
## 预处理概念

所谓的预处理技术，最初也是由MySQL提出的一种减轻服务器压力的一种技术！

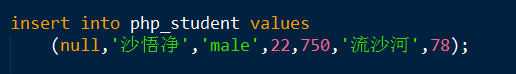
先来看看传统的SQL语句的执行过程：

1. 在客户端准备sql语句
2. 发送sql语句到MySQL服务器
3. 在MySQL服务器执行该sql语句
4. 服务器将执行结果返回给客户端

但是，如果一个脚本中需要插入多条记录的时候，可能会连续执行如下的数据库操作：







……

上面的sql语句当然都可以正常的执行，但是采用这种方法执行sql语句效率并不高！

因为每次执行的sql语句的命令和结构都是一样的，只是其中的数据不同，但是，由于采用传统的执行方式，每次都必须传输整条sql语句，而且对于MySQL服务器而言，得到的每一次的命令都是全新，都是根据sql语句从0解析并执行！

**思考：**

如何优化？

在MySQL中，就是采用预处理机制进行优化的！

**预处理的基本策略：**

将sql语句强制一分为二：

第一部分为前面相同的命令和结构部分

第二部分为后面可变的数据部分

在执行sql语句的时候，首先将前面相同的命令和结构部分发送给MySQL服务器，让MySQL服务器事先进行一次预处理（此时并没有真正的执行sql语句），而为了保证sql语句的结构完整性，在第一次发送sql语句的时候将其中可变的数据部分都用一个数据占位符来表示！比如问号？就是常见的数据占位符！

## MySQL中的预处理

### 不带参数的预处理

#### 第一步：准备预处理语句

语法形式：

**prepare 语句名称 from “预处理的sql语句”;**

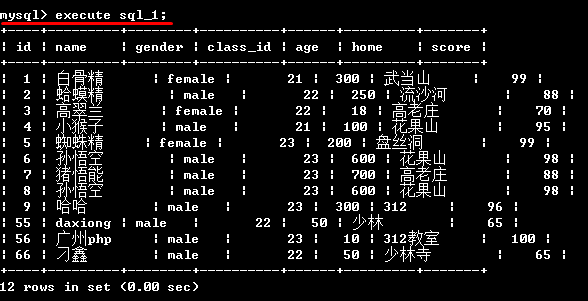
其中语句名称就是给后面需要预处理的sql语句起一个名字！



#### 第二步：执行预处理语句

语法形式：

**execute 语句名称;**



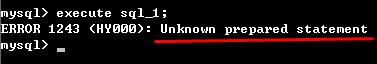
#### 第三步：删除预处理

语法形式：

**drop prepare 语句名称;**



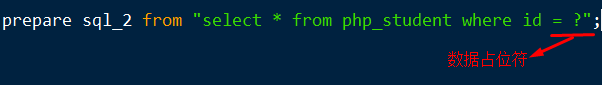
删除之后，就不能再执行预处理语句了！



### 需要传参的预处理

#### 第一步：准备预处理语句

和前面的语法是一样的，只是其中可变的数据部分用一个数据占位符来表示：





#### 第二步：定义参数变量

语法形式：

**set @变量名 = 值;**

注意：这里的@是在MySQL中定义变量的一种语法形式（类比php中的$符号）



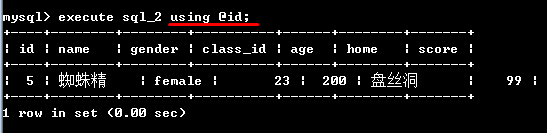


#### 第三步：传递参数变量并执行预处理语句

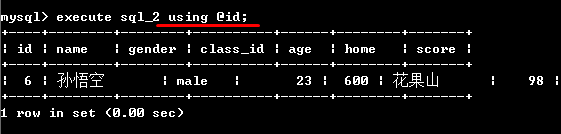
语法形式：

**execute 语句名称 using 参数变量;**

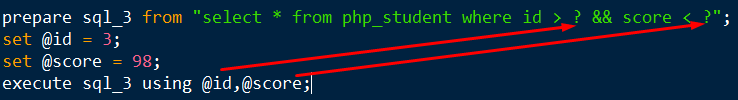
**注意：**using后面只能是参数变量不能是一个常量！



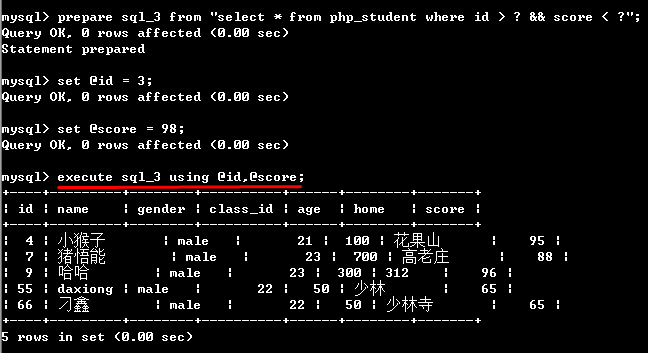
如果想查询其他的记录，只需要修改@id的值就行了！



如果数据占位符不止一个，就按数据占位符的顺序传参就行了：



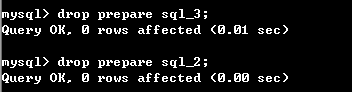
执行结果如下：



#### 第四步：删除预处理

语法形式：

**drop prepare 语句名称;**



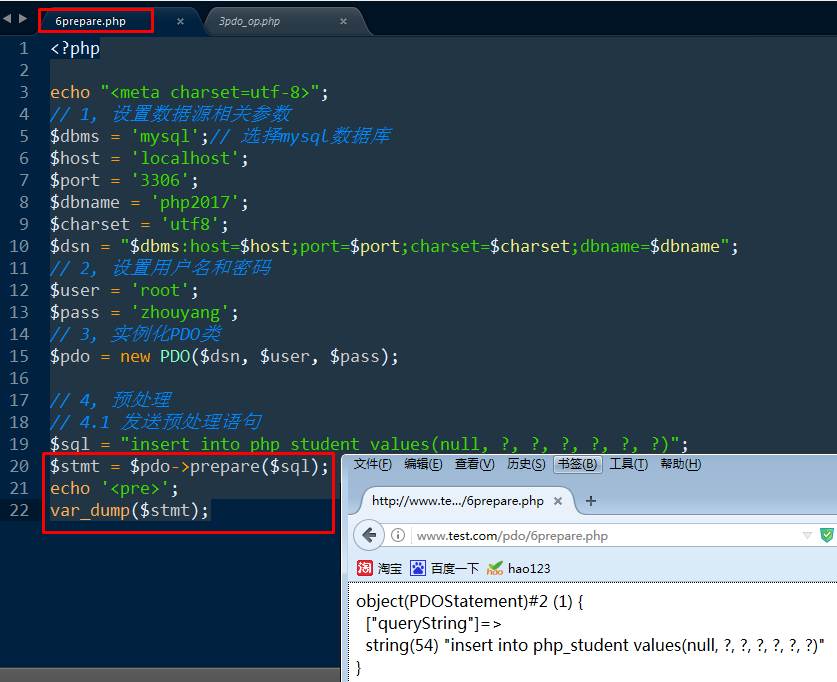
# 第6讲 PDO中的预处理

基本的原理和步骤和前面都是一样的，只不过就是把前面所有命令行的语法封装成了PDO对象的几个公开的方法而已！

## 第一步：发送预处理语句

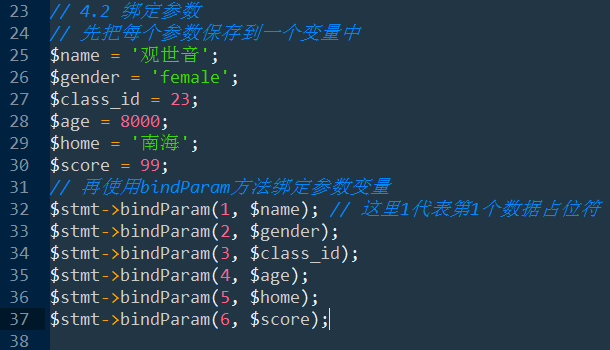
此时，我们需要调用pdo对象的prepare方法，得到一个PDOStatement结果对象！

PDO::prepare — Prepares a statement for execution and returns a statement object



## 第二步：绑定参数

这个时候，需要调用PDOStatement对象中的bindParam方法：



**注意：**

使用bindParam方法绑定的时候不能使用常量直接绑定，因为此时绑定的数据与数据占位符之间是一种引用传递，只能用变量！

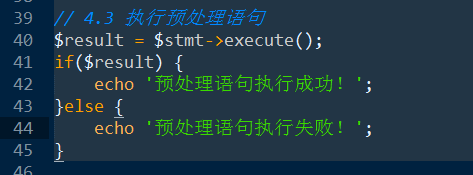
还有一个绑定参数的方法：

bindValue，与bindParam的区别是，采用bindValue进行绑定的时候，与数据占位符之间是一种值传递！

## 第三步：执行预处理语句

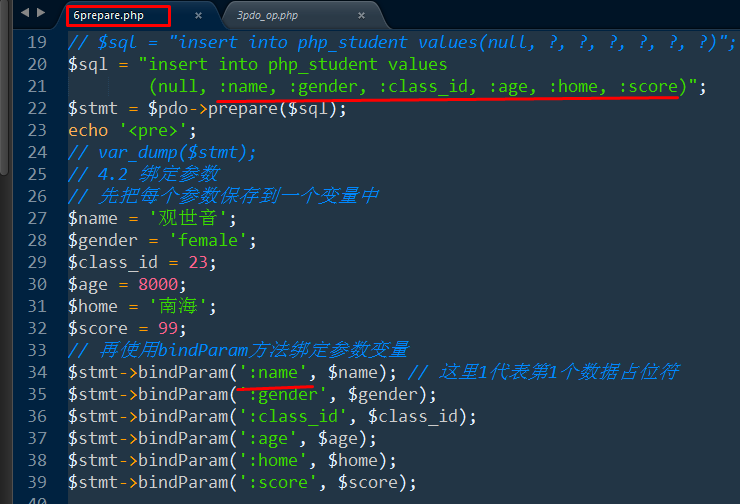


此时，我们需要调用PDOStatement对象中的execute方法，如果执行成功，就返回true，如果执行失败就返回false！



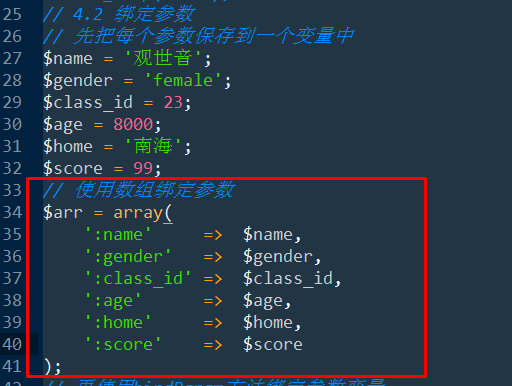
## 绑定参数的其他方法

### 使用 :变量名 作为数据占位符

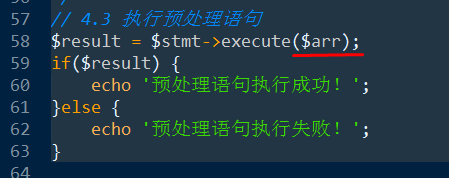


### 使用数组绑定参数

就是使用数组以键值对的形式，实现保存数据占位符与值的对应关系：



然后在调用execute方法的时候，将这个数组作为参数进行传递即可：

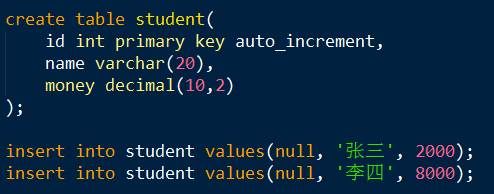


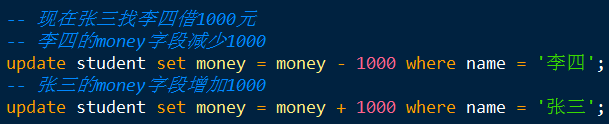
# 第7讲 PDO事务处理

## 事务的概念

什么是事务？

比如：借钱！





但是，有一个问题，以上的另个update语句毕竟是两个独立的sql语句，MySQL服务器需要独立的执行两次！但是从业务逻辑上看，这两个sql语句必须全部执行成功整个业务逻辑才算是执行成功！只要有一个执行失败，就不符合正常的业务逻辑！

所以，我们可以将上面的两个操作（两个update）认为是一个事务，其中有任意一条语句执行失败，我们就认为整个事务执行失败！应该回滚“恢复”到最初的状态！

**注意：**目前只有InnoDB引擎才支持事务处理

## MySQL的事务的处理

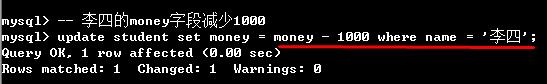
### 开启事务

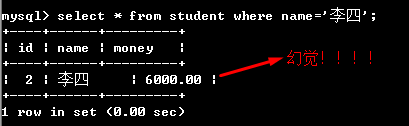
语法形式：

**start transaction;**



然后可以执行sql语句：

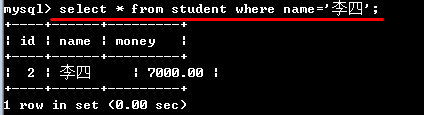




**注意：**

当我们开启一个事务的时候，我们对sql的操作都发生在内存中，但是没有真正的反馈到数据库磁盘的文件中！

我们可以开启另外一个客户端进行查看：



所以，如果此时李四反悔了，此时可以进行回滚操作！

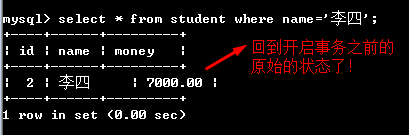
### 回滚事务

语法形式：

**rollback;**

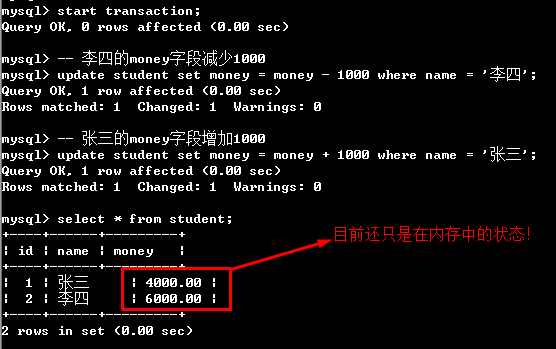
所谓的事务的回滚，就是恢复到事务开启之前的最原始的状态！





**注意：**

回滚操作会自动的关闭一个事务，如果想再次执行事务，需要重新开启事务！



### 执行事务

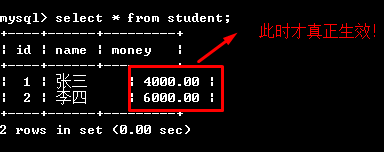
语法形式：

**commit;**



到此为止，整个事务处理完毕！

此时，通过另外一个客户端进行查看：



思考：

以前我们在执行sql语句的时候，为什么都可以直接生效呢？

## 事务的基本原理

普通的执行，之所以是立即执行并生效，因为默认的，MySQL对sql语句的执行是自动提交的！

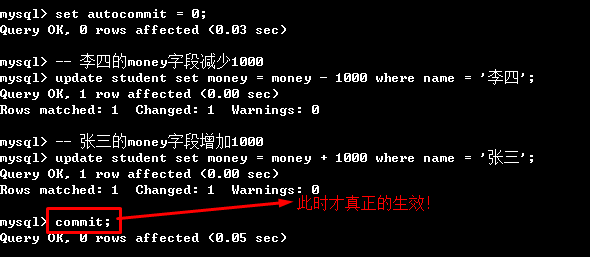
所以，开启一个事务的本质，就是关闭了以前的自动提交的功能，而是由用户手动提交（利用commit语句）！

所以，还有一种开启事务的方式，就是将以前自动提交的功能关闭：

语法形式为：

**set autocommit = 0;**





此时，每次执行sql语句都需要手动的进行提交！

探讨两种开启事务方式的不同：

1. 通过start transaction开启：commit之后相当于上一个事务执行完毕并关闭了事务（以后是sql语句的执行还是默认自动提交）
2. set autocommit=0：关闭了自动提交的功能，以后所有的sql语句都要手动的commit才能够生效！

推荐使用第一种开启事务的方式！

总结事务的步骤：

1. 开启事务
2. 如果执行成功，就提交commit
3. 如果有任何一条sql语句执行失败，则回滚rollback！

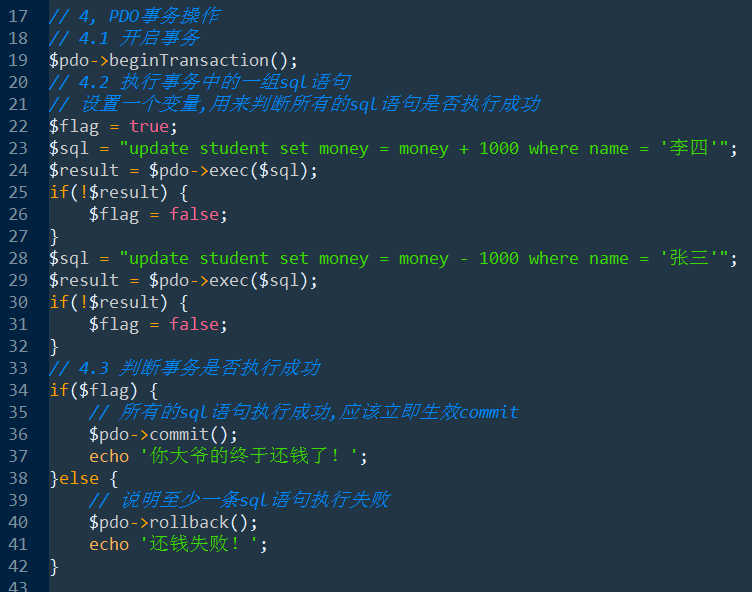
## PDO中的事务处理

基本原理和步骤其实都是一样的，就是调用PDO对象的三个方法：

开启事务：beginTransaction

回滚操作：rollBack

执行操作：commit



# 第8讲 PDO的相关属性（参数）

虽然PDO所有的属性都不是public型，但是我们还是可以通过两个方法来设置和查看PDO的属性：

**setAttribute：**设置属性

**getAttribute：**获取数组的值

PDO中的属性有很多，常见的最重要的有以下的几个：

PDO::ATTR\_AUTOCOMMIT：自动提交

只有两个值：0或1

0代表不开启自动提交，手动提交（调用commit方法）

1代表开启自动提交，自动实现事务处理

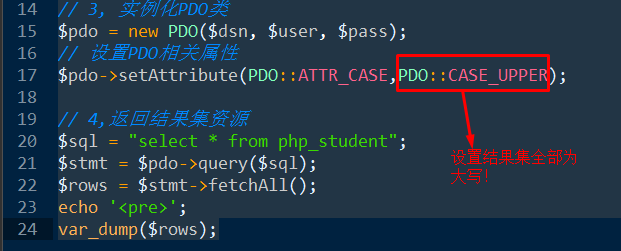
PDO::ATTR\_CASE：返回结果集的大小写（重要）

有三个值可以设置：

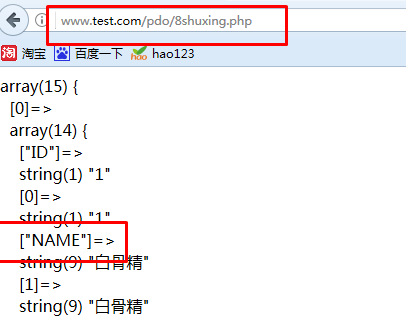
PDO::CASE\_LOWER：返回的结果集全部小写！

PDO::CASE\_UPPER：返回的结果集全部大写！

PDO::CASE\_NATURAL：返回的结果集正常（默认值，原来数据库中什么样现在返回的还是什么样）



执行结果如下：



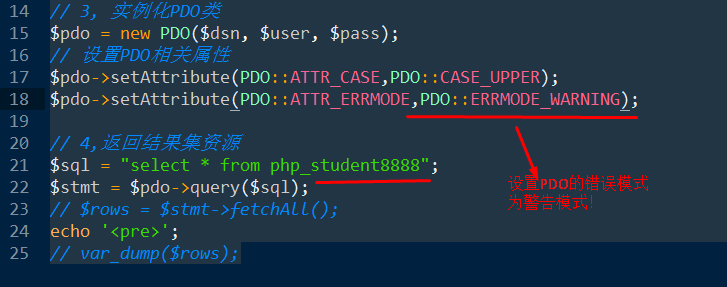
PDO::ATTR\_ERRMODE：返回的错误模式（重要）

有三个值可以设置：

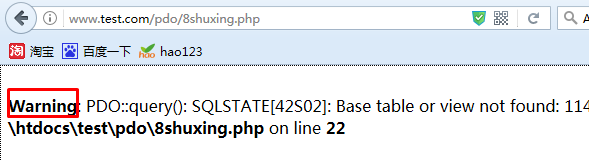
PDO::ERRMODE\_SILENT：静默模式（默认）

PDO::ERRMODE\_WARNING：警告模式

PDO::ERRMODE\_EXCEPTION：异常模式



执行结果如下：



# 第9讲 PDO异常处理

## 异常的概念

异常，其实就是错误，不过是在面向对象编程语言中的一种典型的错误处理方式！所谓的异常，就是把错误信息，放到一个 对象中！

但是，异常的发生有一个前提，就是必须保证语法是正确的！

## 异常的处理

首先要发生异常！

所谓的发生，指的是抛出一个异常！

然后是处理异常！

所谓的处理，就是一旦异常出现，要得到并处理这个异常，通常有两个阶段组成：监听和捕获！

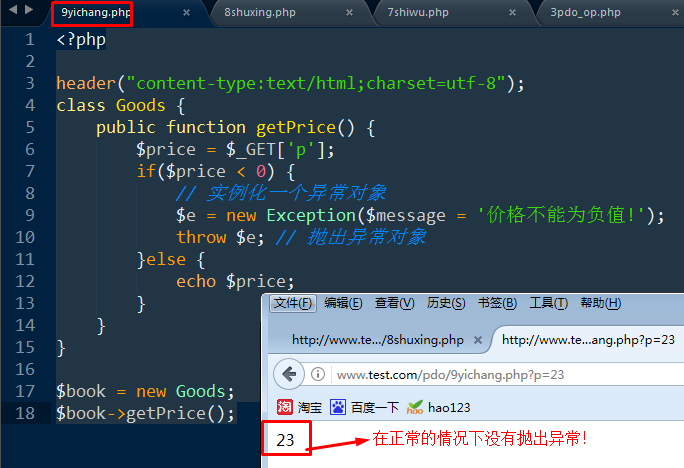
### 抛出异常，throw

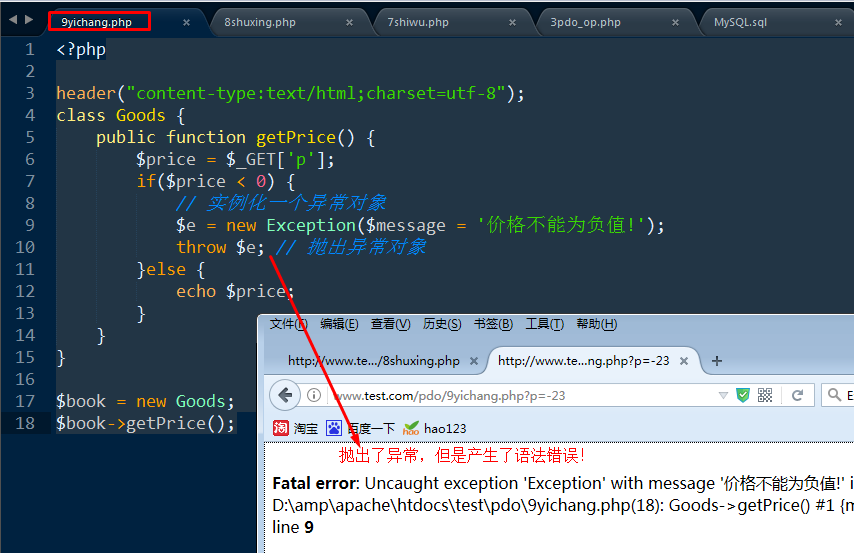
抛出异常其实就是利用throw关键字来完成的，语法形式为：

throw 异常;

**注意：**

这里的异常，其实就是一个异常对象，这个对象必须是由系统预定义的异常类（类名为Exception）或者其子类实例化出来的！





报错的原因是此时只是抛出了一个异常，并没有进行监听和捕获！（类比抛绣球招亲）

### 异常监听，try

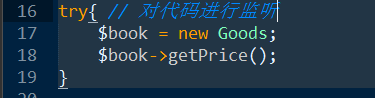
利用关键字try可以监听一段有可能出现异常的代码内，来监听是否发生了异常！

语法为：

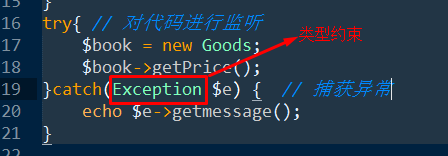
try{

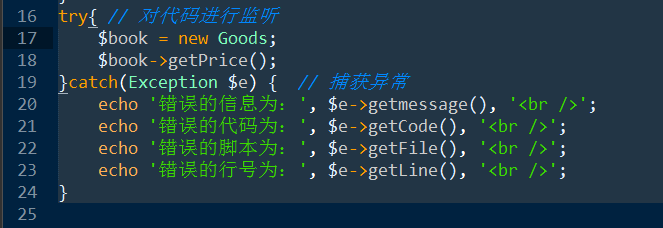
被监听的代码段;

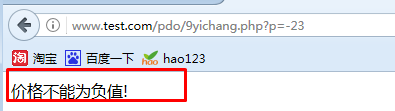
}



### 捕获，catch







这里的$e就是前面代码中抛出的那个异常对象！

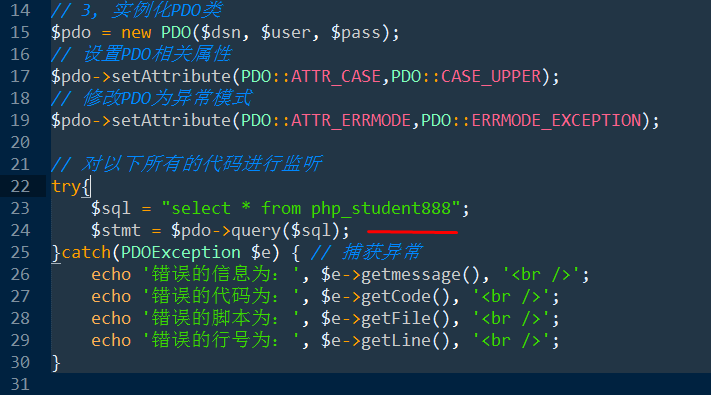
## PDO中处理错误的方式

一共提供了三种错误处理的方式：

PDO::ERRMODE\_SILENT：静默模式（默认）

PDO::ERRMODE\_WARNING：警告模式

PDO::ERRMODE\_EXCEPTION：异常模式



执行结果为：

