**oop\_01**

**1.什么是对象？**

对象:自然间形形色色存在的可以看的见东西

在代码里面:由代码执行出来的效果都是对象

**2.什么是面向对象？**

oop（面向对象编程）他是object oritended programming 缩写，具有灵活性，可维护性，能为程序员开启一扇大门。

软件工程达到的三个目标:重用性，灵活性，可扩展性

面向对象的特点: **封装，继承，多态**

特性:**对象行为 对象状态 对象标识** 相同对象行为和对象状态的具有**不同标识**

**3.什么是类？**

具有属性和方法对象的抽象

**4.类的定义**

具有相同**属性**和**行为**对象的抽象

**成员属性 成员方法** 或者 **成员变量 成员函数**

成员属性不可以是带运算符的表达式、变量、方法或函数调用

常用属性修饰符：public、protected、private、static、var(过时)

常用方法修饰符：public、protected、private、static、abstract、final

类的的里面只有成员属性和成员方法

class 类名{

//成员属性

修饰符（必须加）$属性名=属性值；(属性值也可以不加)

//成员方法

修饰符function 方法名（参数）{

主体

return 返回值

}

}

**5.类和对象的关系？**

类的实例化就是对象

**6.实例化**

$变量名=**new** 类名()；//创建出一个对象$变量名

**7.->**

当调用成员属性和成员方法的时候，我们是通过->调用的

**8.$this**

**特殊的对象引用** 代表的是本对象，而且**只能在成员方法里面访问**对象的属性和对象的方法

**9.构造和析构**

this对象外部调用消耗的资源比较多，用构造方法可以降低系统资源。

**构造方法:**当我实例化对象的时候，会自动调用这个构造函数 ，目的是初始化属性值

function **\_\_construct()**{

函数主体

}

**析构方法：**对象被**销毁前**，会自动会调用析构方法

function **\_\_destruct()**{

函数主体

}

当对象被销毁之前，自动调用析构函数(注意析构方法是**没有任何参数**的)

10.同一个类可以new出多个不同的对象，而且所有的对象都是**独立**里面的成员属性和方法**不会共享**。

11.model类（用对象操作学生表 显示学生信息）

class Model{

//成员属性

public $tablename;//表名

public $link;//连接资源

//成员方法

public function \_\_construct($tablename){

$this->tablename=$tablename;

$this->link=mysqli\_connect(HOST,USER,PASS)or die('数据库连接失败');

mysqli\_select\_db($this->link,DBNAME);

mysqli\_set\_charset($this->link,'utf8');

}

//获取学生信息表的数据

public function select(){

$sql="select \* from {$this->tablename}";

$result=mysqli\_query($this->link,$sql);

$a=[];

while($row=mysqli\_fetch\_assoc($result)){

// var\_dump($row);

$a[]=$row;

}

return $a;

}

//析构方法

public function \_\_destruct(){

mysqli\_close($this->link);

}

}

**oop\_02**

**1.封装**

封装:就是在类的成员方法和成员属性前面，加上类似于 public private protected 修饰符，目的是尽量的**隐藏对象的内部细节,目的是达到访问控制**（记住不是拒绝访问）

**<1>修饰符:**

public 公有的

private 私有

protected 受保护的

**<2>在类的外部的访问范围:**

在类的外部可以访问公有属性和方法

在类的外部不可以访问私有属性和方法

在类的外部不可以访问受保护属性和方法

**在类的内部的访问范围:**

在类的内部都可以访问公有的，私有的，受保护的属性和方法

**<3>私有属性提供公有的方法** setName getName

Eg:

public function setName($name ) {

$this->name=$name;

}

public function getName(){

return $this->name;

}

<4>同一个类的对象即使不是同一个实例也可以互相访问对方的私有与受保护成员。这是由于在这些对象的内部具体实现的细节都是已知的。

**2.魔术方法：**

\_\_get 在非公有属性**输出**的时候，会自动的调用\_\_get方法

Function \_\_get(属性名){......}

\_\_set 在非公有属性**赋值**的时候，会自动的调用\_\_set方法

Function \_\_set(属性名,属性值){......}

\_\_unset **销毁**一个对象的私有或受保护的属性时，自动调用此方法，并将属性名以第一个参数传进去

Function \_\_unset(属性名){......}

private function \_\_unset($propertyName) {

if($propertyName == "name") **return**;

unset($this->$propertyName);

}

\_\_isset 判断一个对象的私有或受保护的属性**是否存在**时，会自动调用此方法

Function \_\_isset(属性名){......}

private function \_\_isset($propertyName) { return isset($this->$propertyName); }

**3.补充：**

1）**取得刚刚插入数据的id**  mysqli\_insert\_id($link)

2）$sql="desc {$this->tablename}";

**oop\_03**

**1.继承**

类里面，子类继承过来的是所有属性和方法

Extends class Student **extends** Person{。。。。}

单继承类 一个子类只能有一个父类，一个父类可以有多个子类，一个子类不可以有多个父类

2.在子类里面不可以访问私有的属性和方法，可以访问公有属性和方法，可以访问受保护的属性和方法

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Public** | **Protected** | **Private** |
| 类的外部 | √ | × | × |
| 类的内部 | √ | √ | √ |
| 子类 | √ | √ | × |

**覆盖** 在子类中定义一个和父类相同方法相同属性，父类的此方法和属性会被覆盖

**扩展** 在子类里面定义一个新增加的方法或者属性，就叫做扩展

**parent::**父类的被覆盖的方法名（）来访问这个被覆盖的父方法，在子类中使用。

parent::\_\_construt();

parent::fun();

**3.final**

只能修饰**类和方法，不能修饰属性，**因为final是常量的意思。

修饰的类**不能被子类继承**。

修饰的方法**不能被子类覆盖**（重用）

**4.static**

修饰成员**方法和属性** 也叫**静态方法**和**静态属性**

public static $count;

public static function getCount(){。。。。}

就是在类的外部**不去实例化**的时候，取到对象的数据

**类名::$静态属性**

**类名::静态方法**

echo MyClass::getCount();

在类的方法中，**self::** 代替$this 去访问以static修饰属性和方法

**self::$静态属性**

**self::静态方法**

静态方法中不可以使用非静态的内容；

若没有出现$this的调用，默认此方法为静态方法。

静态属性是共享的。也就是new很多对象也是共用一个属性。

静态属性不能通过一个类已实例化的对象来访问（但静态方法可以）。

**5.const 用来定义常量**

const是一个在**类中定义常量**的关键字，我们都知道在PHP中定义常量使用的是”define()”这个函数，但是在类里面定义常量使用的是”const”这个关键字

const**只能修饰的成员属性**（常量属性），其访问方式和static修饰的成员属性访问的方式差不多，也是使用”类名”，在方法里面使用”self”关键字。但是不用使用”$”符号，也不能使用对象来访问。

const CONSTANT =‘constant value’; //定义

echo self::CONSTANT; //类内部访问

echo ClassName::CONSTANT; //类外部访问

**6.instanceof**

用来检测实例对象是否属于某一个类

由子类生成的对象也是属于父类的，返回true

echo $p instanceof Student; //结果为false

1. model类的 修改 获取单条数据 增删改差 表格增删改查
2. 补充

<1>在**类的成员方法**里面，可以用 ->（对象运算符）：**$this->**property（其中 property 是该属性名）这种方式来访问**非静态属性**。**静态属性**则是用 ::（双冒号）：**self::$**property来访问。

<2>范围解析操作符 :: 用于访问静态成员，类常量，还可以用于覆盖类中的属性和方法。

self，parent 和 static这三个特殊的关键字是用于在类定义的内部对其属性或方法进行访问的。

**oop\_04**

**1.克隆 clone**

普通的对象 类似于数组，常量等等，**可以全部克隆**

特殊的对象 一个对象含有**资源** 特殊对象克隆的时候 **资源不能够被克隆过去，**eg：读取文件的内容

克隆出来的对象并不是同一个对象

**2.\_\_clone**

当**对象克隆**的时候，会自动调用\_\_clone()方法

**3.\_\_toString**

当**输出对象**的时候，会自动调用\_\_toString()这个方法，快速获取对象的字符串表示

**返回字符串**

**括号里无任何参数**

**Eg:** $m = new A; echo $m;

**4.\_\_call**

当输出一个**不存在的成员方法**，自动的调用\_\_call()方法

\_\_call(不存在方法名,不存在方法名参数（数组）); \_\_call($functionname,$array)

public function \_\_call($name,array $array){

var\_dump($name);

var\_dump($array);

echo '方法不存在';

}

**5.\_\_autoload**

当实例化一个**不存在的类**的时候，自动的调用\_\_autoload();

function \_\_autoload($name){

if(file\_exists("./a/{$name}.class.php")){

require("./a/{$name}.class.php");

echo '<br />'.$name.'<br />';

}elseif(file\_exists("./b/{$name}.class.php")){

require("./b/{$name}.class.php");

}else{

echo "准备实例化的{$name}类不存在！";

}

}

**6.serialize** 串行化（Serialization）

在网络中传输对象数据的时候,将对象串行化成二进制串

保存文件的时候，将字符串写入文件

串行化一个对象将会保存对象的所有属性变量和类名信息，但是**不保存对象的方法**。

Eg：O:1:"A":1:{s:5:"shuju";s:9:"金瓶梅";}

**7.unserialize**

反串行化 PHP串行化可以把变量包括对象，转化成连续bytes数据，你可以将串行化后的变量存在一个文件里或在网络上传输，然后再反串行化还原为原来的数据。

为什么要使用串行化:

1）数据通信，不同的站点之间的数据通信，例如A站点去取B站点的某些数据，返回数据用serialize后作为字符串传输，接收方unserialize一下即可还原成实际的数据

2）数据存储，同样的，也是为了作为字符串，将变量的值存储到 文件/数据库 里面，当需要用到的时候取出来处理一下即可

总结一下：serialize的主要作用是为了**数据共享**和**传输**，将一些数据共享给其他文件或者站点

**8.\_\_sleep**

当串行化的时候自动的调用\_\_sleep()方法

**返回的必须是数组形式**

目的是**实现资源类型属性的关闭释放**等操作。

Eg:

public function \_\_sleep(){

$this->money = $this->money +10000;

return array('a'=>'name','b'=>'age','c'=>'money');

} //name/age/money都是**变量名，不能写成$this->name的形式**

**9.\_\_wakeup**

当反串行化的时候自动调用\_\_wakeup()方法

目的是实现资源属性的打开（sleep方法关闭的资源），即再次初始化。

**10.类型约束**

可以使用的类型是：数组或对象

Eg： public function test\_array(**array** $input\_array) {

print\_r($input\_array);

}

**oop\_05**

**1. 抽象类：**

当类中有一个方法，**没有方法体**，也就是没有花括号，直接分号结束。

如 public abstract function fun();

这种方法我们叫抽象方法，必须**使用关键字abstract定义**，

**包含这种方法的类必须是抽象类也要使用关键字abstract加以声明**.

**抽象类的特点：**

不能实例化，也就new成对象

若想使用抽象类，就必须定义一个类去**继承**这个抽象类，并定义**覆盖**父类的抽象方法(实现抽象方法)。

抽象类里面的抽象方法和属性可以加public private protected

可用\_\_construct函数

在抽象类中可以声明正常的非抽象的方法

含有抽象方法的类肯定是抽象类，但是不是所有的抽象类都必须包含抽象方法。

**2. 接口:**

接口中**只能有常量与抽象方法**。

格式：假如一个抽象类中所有的方法都是抽象的，那么我们可以使用另外一种方式定义：接口

接口使用关键字**interface**来定义，

interface 接口名{

[常量定义]

[抽象方法定义] //注意不要有abstract关键字

}

**接口的特点：**

接口里**只能有常量和抽象方法**

**不能实例化**

实现接口用**implements**，不能用子类取继承接口

在接口里的**抽象方法只能是用public**去修饰

**不能用构造函数**\_\_construct()

当一个类在继承了一个接口后，它必须实现即**覆盖该接口的所有方法**才可以实例化使用，否则即为抽象类。

实现方式：class 类名 implements 接口名，接口名2{

实现体....

}

**3.抽象类和接口的区别？**

（1）对接口的使用方式是通过**关键字implements实现**的，而对于抽象类的操作是使用类继承的**关键字extends实现**的，使用时要特别注意。

（2）接口没有**数据成员**，但是抽象类有数据成员，抽象类可以实现数据的封装。

（3）接口没有**构造函数**，抽象类可以有构造函数。

（4）接口中的方法都是**public类型**，而抽象类中的方法可以**使用private、protected或public**来修饰,目的是进行数据的封装

**4.多态**

\*多态(使用方式)：**对于同一个方法，传入不同对象，实现了不同的效果，**这个就是多态的意思，

需要使用的技术：继承或实现，方法的覆盖（重写）。

Eg：

interface PCI{

public function start();

public function stop();

}

//定义一个主板类

class MainBorad{

public function running($pci){

$pci->start();//声卡开启

$pci->stop();//声卡的关闭

}

}

//定义一个声卡类

class Sound implements PCI{

public function start(){

echo "声卡开启<br>";

}

public function stop(){

echo "声卡关闭<br>";

}

}

//定义一个网卡类

class NetWork implements PCI{

public function start(){

echo "网卡开启";

}

public function stop(){

echo "网卡关闭";

}

}

//类的实例化

$Sound = new Sound();//声卡对象

$NetWork=new NetWork();//网卡对象

$MainBorad=new MainBorad();//主板类

$MainBorad->running($Sound);//在主板的同一个方法里传入声卡对象

$MainBorad->running($NetWork);//在主板的同一个方法里传入网卡对象

**4.补充**

**$\_SERVER[‘PHP\_SELF’]** 返回当前网页的地址

**Mysqli\_fetch\_row($result)** 返回结果的第一行数据，和SELECT COUNT(\*) ROM USERS结合可以求出对应查询数据的数量。

**oop\_06**

**1.异常处理：**

异常（Exception）处理用于在**指定的错误发生时**改变脚本的正常流程。是PHP5中的一个新的重要特性。

异常处理是一种可扩展、易维护的错误处理统一机制，并提供了一种新的面向对象的错误处理方式。

**异常处理格式：**

try{

使用try去包含可能会发生异常的代码.

一旦出现异常try进行捕获异常，交给catch处理。

抛出异常语句：throw 异常对象。

}catch（异常对象）{

在这里做异常处理。

}

在php5中有一种新的错误处理机制--异常处理：（采用面向对象方式的）

**2. 自定义异常处理**

throw new Exception(“异常信息”)





**3.对象函数**

1. class\_alias — 为类创建一个别名

class\_alias("Stu","Demo"); //为类Stu起来一个别名Demo

$s = new Demo();

2. class\_exists — 检查类是否已定义

先判断stu类是否存在，然后再执行

if(class\_exists("Stu")){

$s2 = new Stu();

}

3.get\_class\_methods — 返回由类的**公共方法名**组成的数组

获取Stu类中的所有方法（公有）

var\_dump(get\_class\_methods("Stu"));

//array(1) { [0]=> string(6) "getAge" [1]=> string(7) "getName" }

4.get\_class\_vars — 返回由类的默认公有属性组成的数组

var\_dump(get\_class\_vars("Stu"));

//array(1) { ["name"]=> string(8) "zhangsan" }

5. get\_class — 返回对象的类名

echo "s2对象的类名：".get\_class($s2);

echo "<hr/>";

6. method\_exists — 检查类的方法是否存在

//判读方法是否存在

if(method\_exists($s2,"getAge")){

echo $s2->getAge();

}

7. property\_exists — 检查对象或类是否具有该属性

if(property\_exists($s2,'name')){

echo $s2->name;

}

8. is\_a — 如果对象属于该类则返回 TRUE

echo "<hr>";

if(is\_a($s2,'Stu')){

echo "对象属于该类";

}

**4.搜索**

（1）封装搜索条件

（2）拼接搜索以后的总条数的sql语句 select sql 主要做分页

（3）搜索结合分页

（4）在url地址追加参数

**oop\_07**

**php设计模式——单例模式**

**单例模式概念**

单例模式是指整个应用中类只有一个对象实例的设计模式。

**单例模式的特点**

1.一个类在整个应用中只有一个实例

2.类必须自行创建这个实例

3.必须自行向整个系统提供这个实例

**PHP中使用单例模式的原因**

PHP大部分操作都是和各种数据库打交道，所以一个应用中会存在大量连接数据库的操作，如果不用单例模式，那每次都要new操作，但是每次new都会消耗大量的内存资源和系统资源，而且每次打开和关闭数据库连接都是对数据库的一种极大考验和浪费。

**例子：**

//初始化一个数据库句柄

$db = new DB(...);

//比如有个应用场景是添加一条用户信息

$db->addUserInfo();......

//然而我们要在另一地方使用这个用户信息，这时要用到数据库句柄资源，可能会这么做

......function test() {

　　　$db = new DB(...);

　　　$db->getUserInfo();

有些朋友也许会说，可以直接使用global关键字！

　　　global $db;

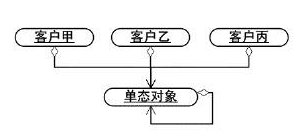
的确global可以解决问题，也起到单例模式的作用，但在OOP中，我们拒绝这种编码。因为global存在安全隐患（**全局变量不受保护**的本质）。

全局变量是面向对象程序员遇到的引发BUG的主要原因之一。这是因为全局变量将类捆绑于特定的环境，破坏了封装。如果新的应用程序无法保证一开始就定义了相同的全局变量，那么一个依赖于全局变量的类就无法从一个应用程序中提取出来并应用到新应用程序中。

确切的讲，单例模式恰恰是对全局变量的一种改进，避免那些存储唯一实例的全局变量污染命名空间。你无法用错误类型的数据覆写一个单例。这种保护在不支持命名空间的PHP版本里尤其重要。因为在PHP中命名冲突会在编译时被捕获，并使脚本停止运行。

单例模式实例：

**先看图：**



上面的对象图中，有一个“单例对象”，而“客户甲”、“客户乙”和“客户丙”是单例对象的三个客户对象。可以看到，所有的客户对象共享一个单例对象。而且从单例对象到自身的连接线可以看出，单例对象持有对自己的引用。

<?php

class User {

//静态变量保存全局实例

private static $\_instance = null;

//私有构造函数，防止外界实例化对象

private function \_\_construct() {

}

//私有克隆函数，防止外办克隆对象

private function \_\_clone() {

}

//静态方法，单例统一访问入口

static public function getInstance() {

if (is\_null ( self::$\_instance ) || isset ( self::$\_instance )) {

self::$\_instance = new self ();

}

return self::$\_instance;

}

public function getName() {

echo 'hello world!';

}

}

?>

**优点：**

1. 改进系统的设计

2. 是对全局变量的一种改进

**缺点：**

1. 难于调试

2. 隐藏的依赖关系

3. 无法用错误类型的数据覆写一个单例

**oop\_07**

**一:Pdo(重点)**

PDO(php data object)就是**操作数据库的方法**，

pdo就是把操作数据库的函数封装成一个**pdo类**

这样，无论你使用什么数据库，都可以通过一致的函数执行查询和获取数据，并能够屏蔽不同数据库之间的差异，使用pdo可以很方便地进行**跨数据库程序的开发**，以及**不同数据库间的移植**，是将来php在数据库处理方面的主要发展方向，

它可以支持mysql,oracle,mssqlsqlite sqserver等多种数据库

**PHP操作MySQL数据库方式有三种：**

**\*1. mysql** 最原始的、纯过程化的 如连接： mysql\_connect(主机名，账号，密码);

mysql\_query();

**2. mysqli** 改进版的、兼容过程化和面向对象化操作

如：连接： mysqli\_connect(主机名，账号，密码，库名) 过程化

new mysqli(主机名，账号，密码，库名)

**\*3. PDO** 通用的，兼容其他数据库 ， **纯面向对象方式**

如： 连接： new PDO(‘跨数据库名字:host=;数据库名字=’,账号，密码);

**选择PDO的原因：**

※ 跨数据库，

※ 执行效率高,

※ 带预处理（防sql注入）、

※ 支持事务操作

**PDO的环境配置**：开启支持PDO

在php.ini配置文件中开启（重点）：

**extension=php\_pdo.dll**

**extension=php\_pdo\_mysql.dll**

在使用PDO之前首先要设置PHP.INI文件，使PHP支持PDO。在PHP安装目录下找到PHP.INI文件，PHP.ini中,去掉"**extension=php\_pdo.dll**"前面的";"号,这个时候php就可以支持pdo了。若要连接数据库，还需要去掉与PDO相关的数据库扩展前面的";"号，**extension=php\_pdo\_mysql.dll** 然后重启Apache服务器即可。

注意在php5 这两个配置文件**默认是开启的**

**在PDO操作中涉及到的对象：**

**PDO**、

**PDOStatement**（预处理对象)、

**PDOException**（异常）

**PDO类的方法：pdo对象方法：**

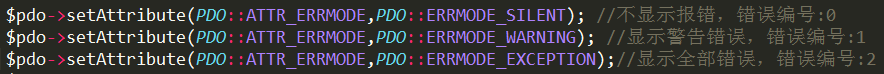
**错误处理模式方法**:

\*PDO::ATTR\_ERRMODE=>错误处理模式：(可以是以下三个) attr\_errmode

PDO::ERRMODE\_SILENT:不报错误（忽略）**(0)**(默认的错误模式 ) errmode\_silent

PDO::ERRMODE\_WARNING:以警告的方式报错**(1)** errmode\_warning

PDO::ERRMODE\_EXCEPTION：以异常的方式报错(推荐使用)。**(2)**  errmode\_exception



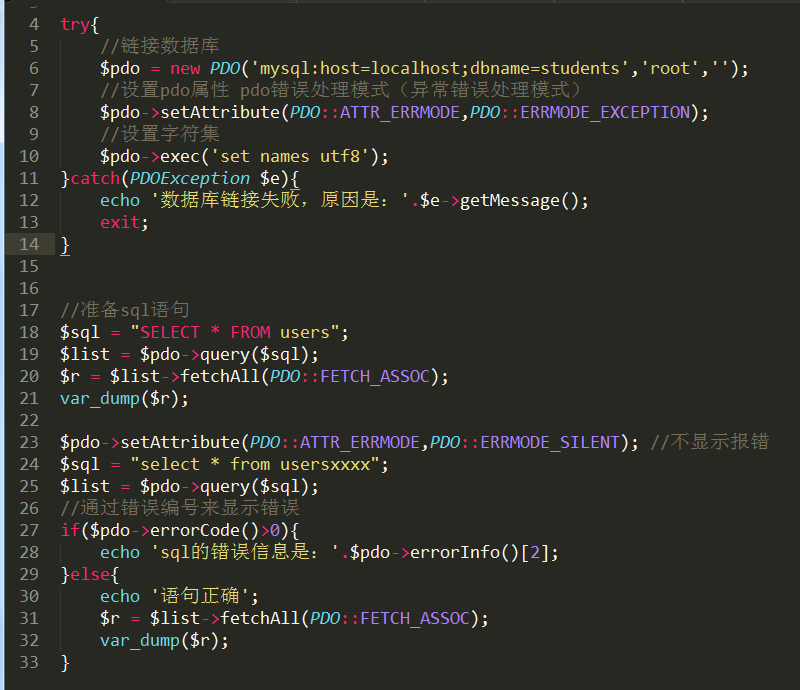
**Pdo连接数据库**

$pdo=newPDO("mysql:host=localhost;dbname=lamp36db","root","");



**开启错误处理模式**

$pdo->setAttribute(PDO::ATTR\_ERRMODE,PDO::ERRMODE\_EXCEPTION); attribute:属性



**关闭数据库**：

$pdo = null;

--------------------------------------------------------

1. query($sql); 用于**执行查询SQL语句**。返回PDOStatement（pdo预处理对象）对象

2. exec($sql); 用于执行**增、删、改**操作，返回**影响行数**；

3. setAttribute(); 设置一个"数据库连接对象"属性。

3.1.getAttribute();获取一样连接数据库对象的属性

4. beginTransaction 开启一个事物（做一个回滚点）

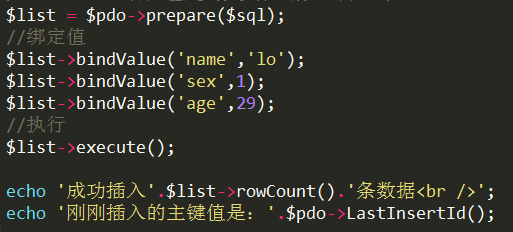
5. commit 提交事务

6. rollBack 事务回滚操作。

7. errorCode 获取错误码

8. errorInfo 获取错误信息

9.lastInsertId 获取刚刚添加的主键值.



10.prepare（$sql） 创建SQL的**预处理**，返回PDOStatement对象

预处理对象PDOStatement对象：大大减少了分析的时间,只**查询一次**,提高查询效率,减少时间。

=============================================

我们可以通过PDO的方法来**获取PDOStatement**：

1.PDO的query（查询sql）方法获取，用于**解析结果集**

2.PDO的prepare(SQL)方法获取，用于**处理参数式sql并执行操作**。

**PDOstatement对象的方法：**

----------------------------------------------------------------

1、fetch() 返回结果集的下一行，结果指针下移，到头返回false 。

参数： PDO::FETCH\_BOTH (default)、：索引加关联数组模式

PDO::FETCH\_ASSOC、 ：关联数组模式

PDO::FETCH\_NUM、 ：索引数组模式

PDO::FETCH\_OBJ、 ：对象模式

PDO::FETCH\_LAZY ：所有模式（SQL语句和对象）

2、fetchAll() 通过一次调用返回所有结果，结果是**以数组形式保存**

参数：PDO::FETCH\_BOTH (default)、

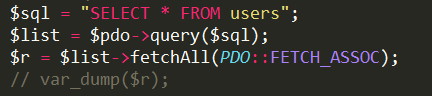
PDO::FETCH\_ASSOC、//关联

PDO::FETCH\_NUM、 //索引

PDO::FETCH\_OBJ、 //对象

PDO::FETCH\_COLUMN表示取指定某一列

如：$rslist = $stmt->fetchAll(PDO::FETCH\_COLUMN,2);**取第三列**



3、execute() 负责执行一个准备好了的预处理语句

4, rowCount() 返回使用增、删、改、查操作语句后**受影响的行总数**

4.1 columnCount() 在结果集中返回列的数目

5. setAttribute() 为一个预处理语句设置属性

6.errorCode() 获取错误码

7. errorInfo() 获取错误信息

8.bindParam() 将参数绑定到相应的**查询占位符**上

bool PDOStatement::bindParam ( mixed $parameter , mixed &$variable [, int $data\_type [, int $length [, mixed $driver\_options ]]] )

其中：

$parameter：占位符名或索引偏移量

&$variable:参数的值，需要按引用传递也就是必须放一个变量

$data\_type:数据类型

PDO::PARAM\_BOOL PDO::PARAM\_NULL

PDO::PARAM\_INT PDO::PARAM\_STR

PDO::PARAM\_LOB PDO::PARAM\_STMT

PDO::PARAM\_INPUT\_OUTPUT

$length：指数据类型的长度

$driver\_options：驱动选项。

9. bindValue() 将一值绑定到对应的**一个参数**中

**防sql语句注入的两种方法:**

**在PDO中参数式的SQL语句有两种(预处理sql)**：

1.insert into stu(id,name) value(?,?); //**？号式**（适合参数少的）

2.insert into stu(id,name) value(:id,:name); // **别名式**(适合参数多的)

**在PDO中为参数式SQL语句赋值有三种：**

1.使用数组

$stmt->execute(array("lamp1404","qq2"));

$stmt->execute(array("id"=>"lamp1404","name"=>"qq2"));

2.使用方法单个赋值

$stmt->bindValue(1,"lamp1901"); bind [baɪnd] ：约束、捆绑

$stmt->bindValue(2,"qq2");

$stmt->execute();

$stmt->bindValue(":id","lamp1901",PDO::PARAM\_STR); //带指定类型

$stmt->bindValue(":name","qq2",PDO::PARAM\_STR);

$stmt->execute();

3. 使用方法绑定变量

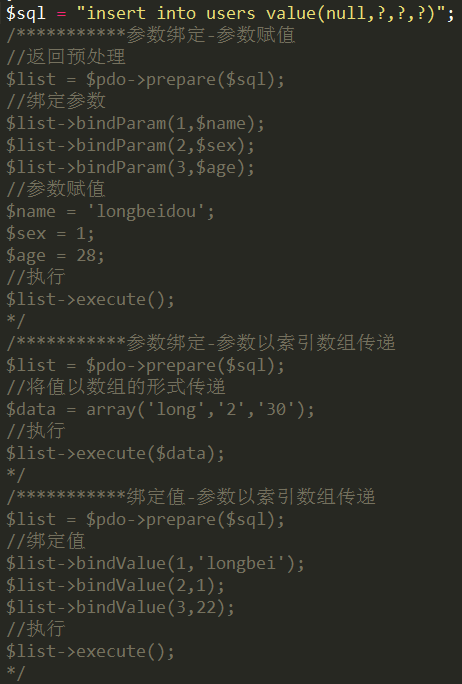
$stmt->bindParam(":id",$id);

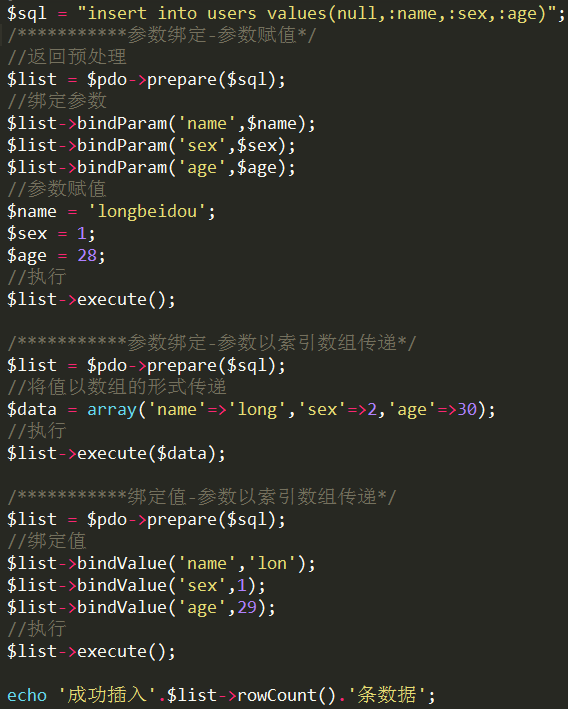
$stmt->bindParam(":name",$name);

$id="lamp1401";

$name="qq2";

$stmt->execute();



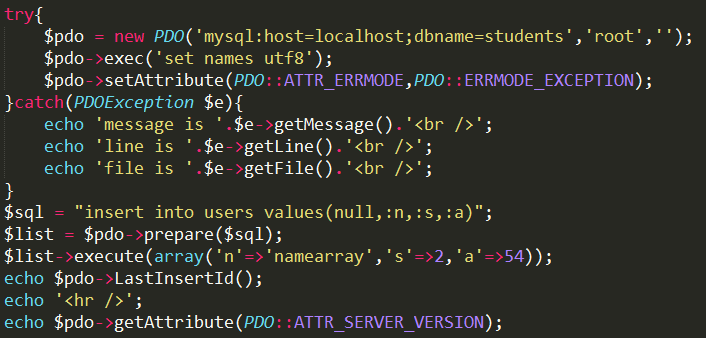


**绑定参数**

1.减少服务器的带宽 只需要发送插入需要的参数,而不是整个的sql语句 提高插入效率

2.参数值在发送后使用的是不同的协议,保证了数据的合法性

二：pdo（防sql注入）操作数据库(增删改查)



三：数据的批量添加(事务处理)

事务：将多条sql操作（增删改）作为一个操作单元，要么都成功，要么都失败。

MySQL对事务的支持:

**被操作的表必须是innoDB类型的表**

（支持事务）

**MySQL常用的表类型**：

MyISAM (非事务)增删改速度快

InnodB （事务型）安全性高

更改表的类型为innoDB类型

mysql> **alter table stu engine=innodb;**

Query OK, 29 rows affected (0.34 sec)

Records: 29 Duplicates: 0 Warnings: 0

mysql> show create table stu\G; //查看表结构

开启一次事务:

**$pdo->beginTransaction(); transaction 事物、业务**

提交一次事务:

**$pdo->commit(); commit 做错事**

回滚一次事务:

**$pdo->rollback();**



**四：补充**

查询数据库语句的执行效率的语句：DESC SELECT \* FROM uses WHERE age=22\G；

