命名空间和PDO

[学习目标 2](#_Toc522379056)

[静态延时绑定 2](#_Toc522379057)

[1、什么是静态延时绑定 2](#_Toc522379058)

[2、实例：静态延时绑定演示 2](#_Toc522379059)

[命名空间概述 4](#_Toc522379060)

[1、什么是命名空间 4](#_Toc522379061)

[2、定义命名空间的要求 4](#_Toc522379062)

[3、实例：定义单个命名空间 5](#_Toc522379063)

[4、实例：定义子命名空间 5](#_Toc522379064)

[使用命名空间 6](#_Toc522379065)

[1、文件系统中访问文件的方式 6](#_Toc522379066)

[2、访问命名空间中元素的方式 6](#_Toc522379067)

[namespace关键字 6](#_Toc522379068)

[命名空间元素的导入和别名 7](#_Toc522379069)

[1、描述 7](#_Toc522379070)

[2、实例：导入命名空间中的类并起别名 7](#_Toc522379071)

[PDO概述 8](#_Toc522379072)

[1、PDO简介 8](#_Toc522379073)

[2、PDO的访问流程图 8](#_Toc522379074)

[3、创建PDO类的对象 9](#_Toc522379075)

[PDO对象常用方法 9](#_Toc522379076)

[1、PDO::exec()方法 9](#_Toc522379077)

[2、PDO::query()方法 10](#_Toc522379078)

[3、PDO::lastInsertId()方法 10](#_Toc522379079)

[4、PDO::setAttribute()方法 11](#_Toc522379080)

[PDOStatement对象常用方法 12](#_Toc522379081)

[1、PDOStatement::fetch()方法 12](#_Toc522379082)

[2、PDOStatement::fetchAll()方法 12](#_Toc522379083)

[3、PDOStatement::rowCount()方法 13](#_Toc522379084)

[PDO错误处理 14](#_Toc522379085)

[1、PDO支持三种错误模式 14](#_Toc522379086)

[2、静默模式(Slient) 14](#_Toc522379087)

[3、警告模式(Warning) 14](#_Toc522379088)

[4、异常模式(Exception) 15](#_Toc522379089)

[SQL语句预处理 17](#_Toc522379090)

[1、SQL语句执行过程 17](#_Toc522379091)

[2、PDO的SQL语句预处理步骤 17](#_Toc522379092)

[（1）PDO完成预处理需要的步骤 17](#_Toc522379093)

[（2）提取相同结构的SQL语句 18](#_Toc522379094)

[（3）预编译相同结构的SQL语句 18](#_Toc522379095)

[（4）给占位符绑定数据 18](#_Toc522379096)

[（5）执行预处理的SQL语句 19](#_Toc522379097)

[3、实例：PDO的SQL语句预处理 19](#_Toc522379098)

# 学习目标

* 命名空间
* PDO对象常用方法
* PDOStatement对象常用方法
* PDO错误处理方式
* SQL语句预处理

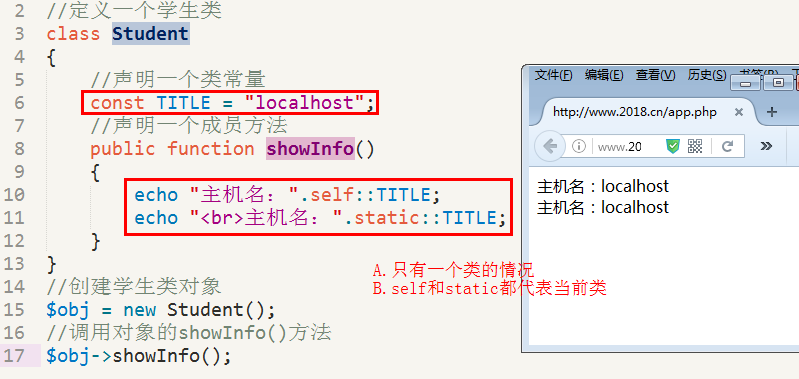
# 静态延时绑定

## 1、什么是静态延时绑定

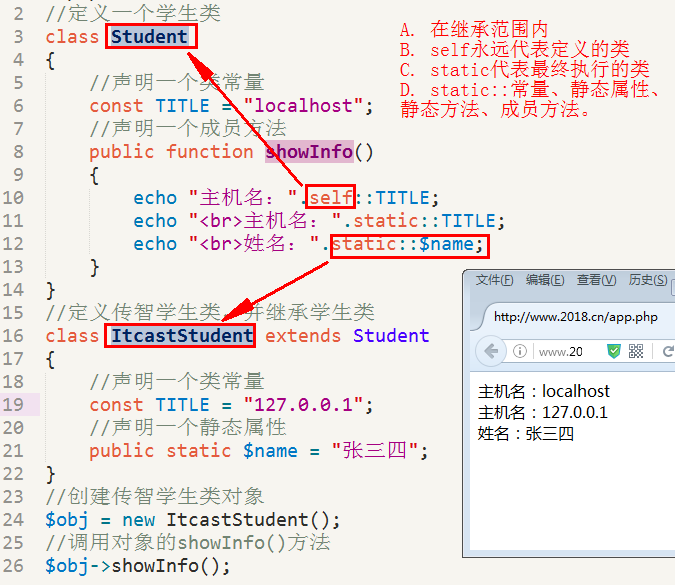
* 自 PHP 5.3.0 起，PHP 增加了一个叫做后期静态绑定的功能，用于在继承范围内引用静态调用的类。
* "后期绑定"的意思是说，static::不再被解析为定义当前方法所在的类，而是在实际运行时计算的。也可以称之为"静态绑定"，因为它可以用于（但不限于）静态方法的调用。
* 我们需要一个在调用执行时才确定当前类的一个特征，就是说将static关键字对某个类的绑定推迟到调用执行时，就叫静态延迟绑定！
* 语法：**static::静态属性，静态方法，成员方法，类常量**

## 2、实例：静态延时绑定演示

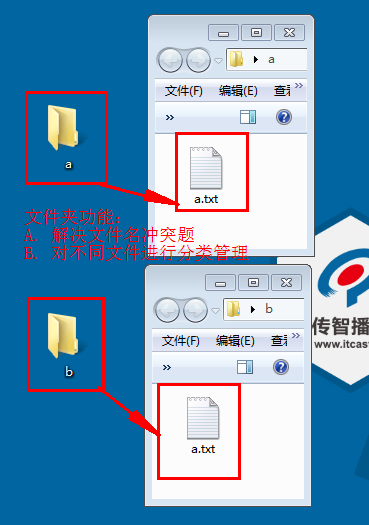
如果只有一个类，self和static都代表当前类



如果在继承范围，self永远代码当前类，static代表最终执行的类



# 命名空间概述



## 1、什么是命名空间

* 从广义上来说，命名空间是一种封装事物的方法。
* 在很多地方都可以见到这种抽象概念。例如，在操作系统中目录用来将相关文件分组，对于目录中的文件来说，它就扮演了命名空间的角色。
* 在PHP中，命名空间用来解决在编写类库或应用程序时名称冲突的问题。
* PHP 命名空间提供了一种将相关的类、函数和常量组合到一起的途径。

## 2、定义命名空间的要求

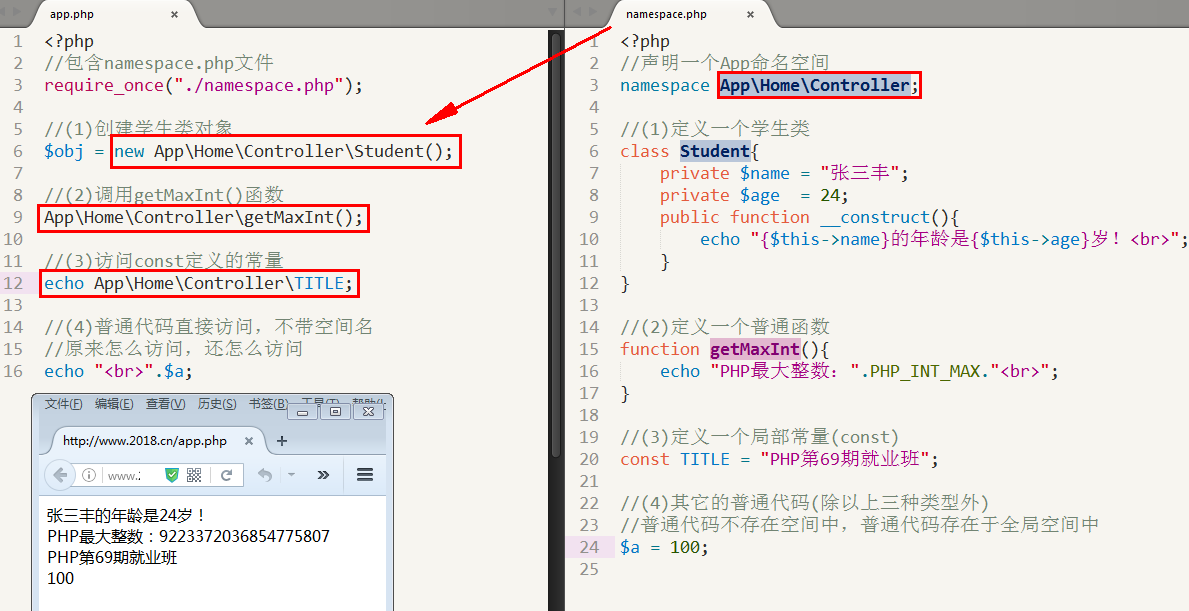
* PHP 在 5.3.0 以后的版本开始支持命名空间。
* 空间中可以包含任意合法的PHP代码，但只有三种代码受命名空间的影响，它们是：类，函数和常量。
* 命名空间通过关键字namespace来声明。
* 命名空间必须是程序脚本的第一条语句。

## 3、实例：定义单个命名空间



## 4、实例：定义子命名空间

* 使用“\”斜线来分割子文件夹。例如：home\controller\a.txt
* 使用“\”斜线来分割子命名空间。例如：$obj = new Home\Controller\Student()



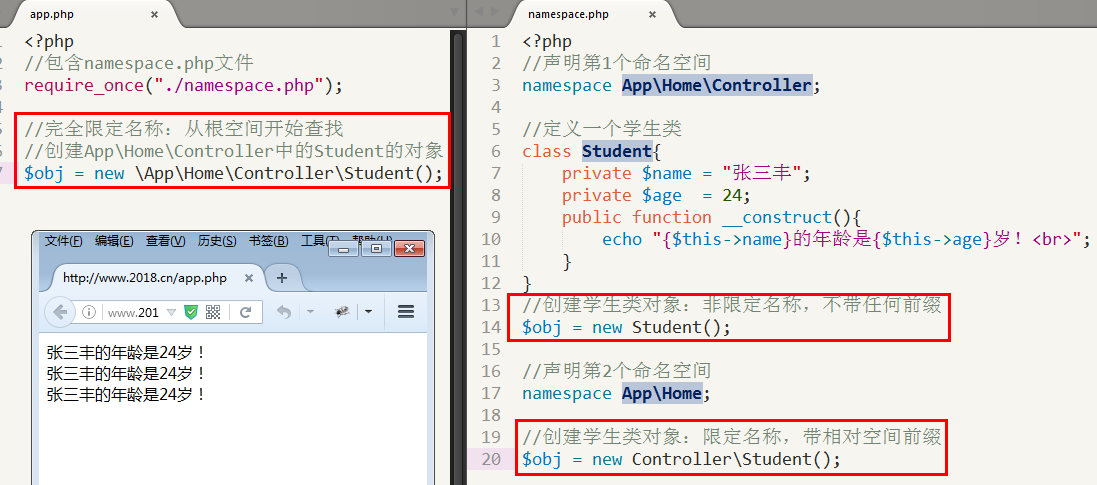
# 使用命名空间

## 1、文件系统中访问文件的方式

* 相对文件名形式，如foo.txt。它会被解析为 curdir/foo.txt。
* 相对路径名形式，如subdir/foo.txt。它会被解析为 curdir/subdir/foo.txt。
* 绝对路径名形式，如/main/foo.txt。它会被解析为/main/foo.txt。

## 2、访问命名空间中元素的方式

* 非限定名称(不带任何前缀)。$obj = new User()，解析为$obj = new current\User()
* 限定名称(带相对空间前缀)。$obj = new Home\User()，解析为$obj = new current\Home\User()
* 完全限定名称(从根空间开始)。$obj = new \Home\User()，解析为$obj = new \Home\User()



# namespace关键字

关键字 namespace 可用来显式访问当前命名空间或子命名空间中的元素。它等价于类中的 self 操作符。

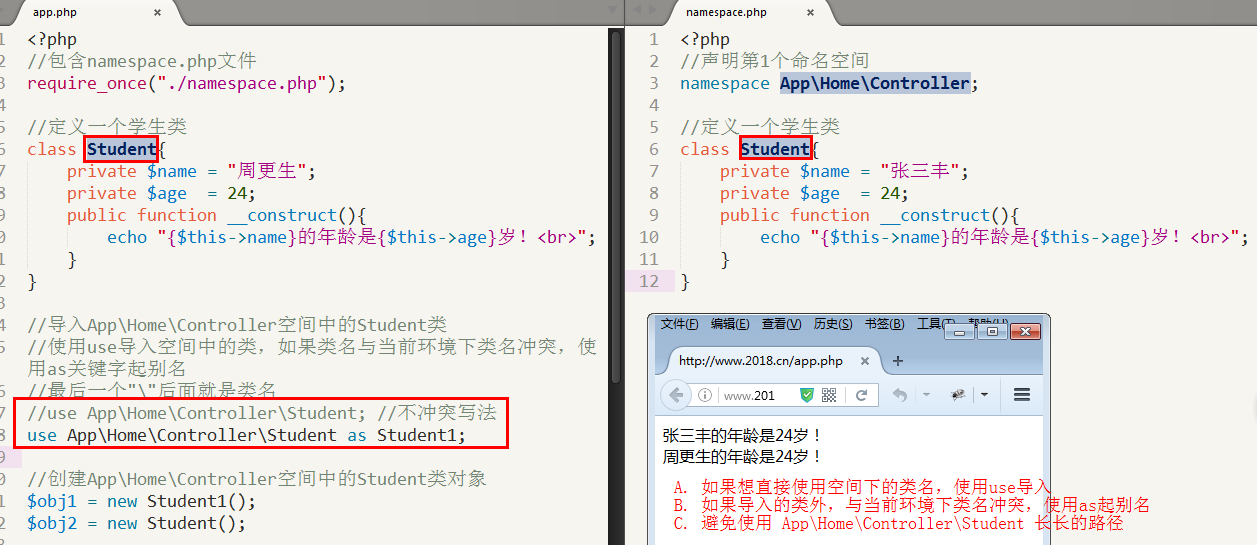


# 命名空间元素的导入和别名

## 1、描述

* 允许通过别名引用或导入外部的完全限定名称，是命名空间的一个重要特征。
* PHP 命名空间支持两种使用别名或导入方式：为类名称使用别名，或为命名空间名称使用别名
* 在PHP中，导入是通过操作符 use 来实现的，别名是通过操作符as来实现的。

## 2、实例：导入命名空间中的类并起别名

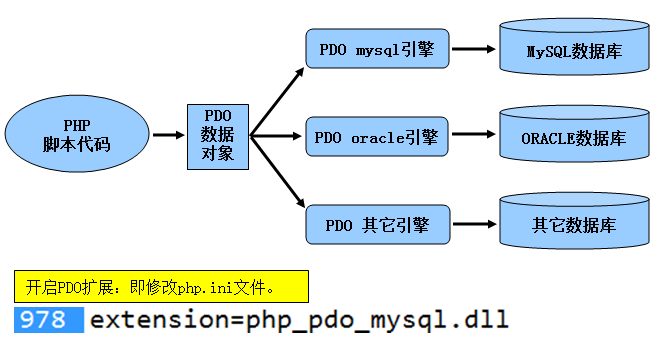


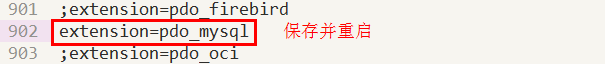
# PDO概述

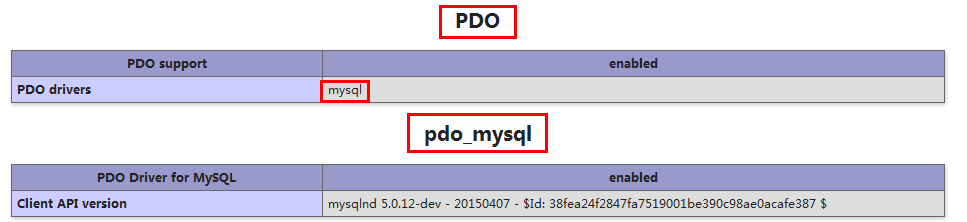
## 1、PDO简介

* PDO是PHP数据对象（PHP Data Object）的缩写。
* PDO扩展为PHP访问不同数据库定义了一个轻量级的、一致性的接口；
* PDO作用是统一各种数据库的访问接口，PDO让跨数据库的使用更具亲和力；
* 有了PDO，您不必再使用mysqli\_\*函数、oci\_\*函数或者mssql\_\*函数，也不必再为它们封装数据库操作类，只需要使用PDO接口中的方法就可以对各种数据库进行操作。
* PDO是一个第三方的类，默认已经集成到PHP中了。

## 2、PDO的访问流程图

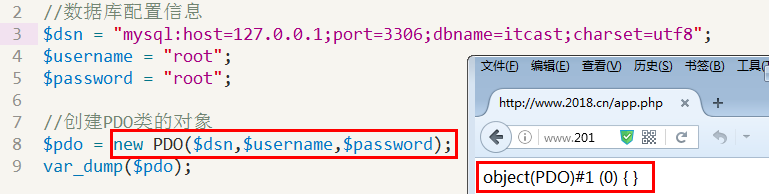






## 3、创建PDO类的对象

* 描述：创建一个表示数据库连接的 PDO 对象。
* 语法：**PDO::\_\_construct ( string $dsn [, string $username [, string $password]] )**
* 参数：
  + $dsn，数据源名称或叫做 DSN，包含了请求连接到数据库的信息。通常，一个 DSN 由 PDO 驱动名、紧随其后的冒号、以及具体 PDO 驱动的连接语法组成。
  + 例如：$dsn = “mysql : host=127.0.0.1; port=3306; dbname=db; charset=utf8”
  + $username，数据库用户名。
  + $password，数据库密码。
* 返回值：成功则返回一个PDO对象。



# PDO对象常用方法

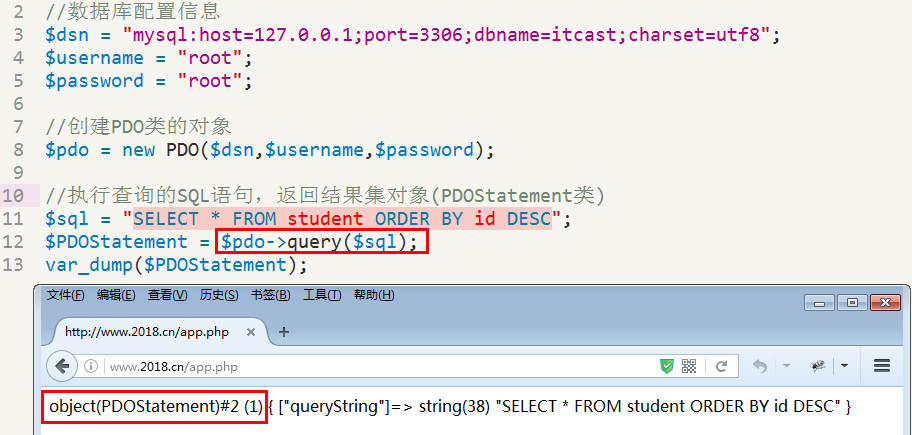
## 1、PDO::exec()方法

* 描述：执行一条 SQL 语句，并返回受影响的行数
* 语法：**int PDO::exec ( string $sql )**
* 参数：$sql要被预处理和执行的 SQL 语句。
* 注意：不会从SELECT语句返回结果。
* 返回：返回受修改或删除 SQL 语句影响的行数。如果没有受影响的行，则返回 0。



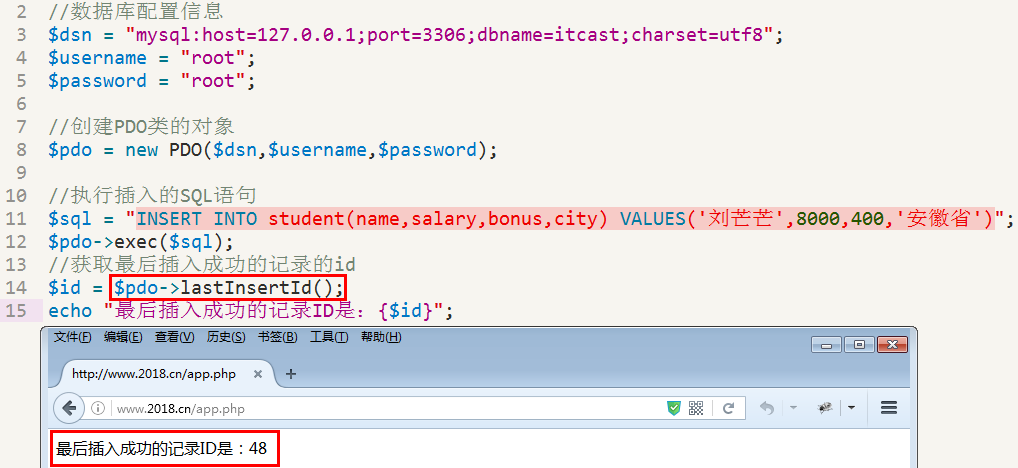
## 2、PDO::query()方法

* 描述：执行一条SQL语句，返回一个结果集对象( PDOStatement )。
* 语法：**public PDOStatement PDO::query ( string $statement )**
* 提示：主要用于SELECT、SHOW语句。
* 返回：执行成功返回PDOstatement对象，执行失败返回FALSE。



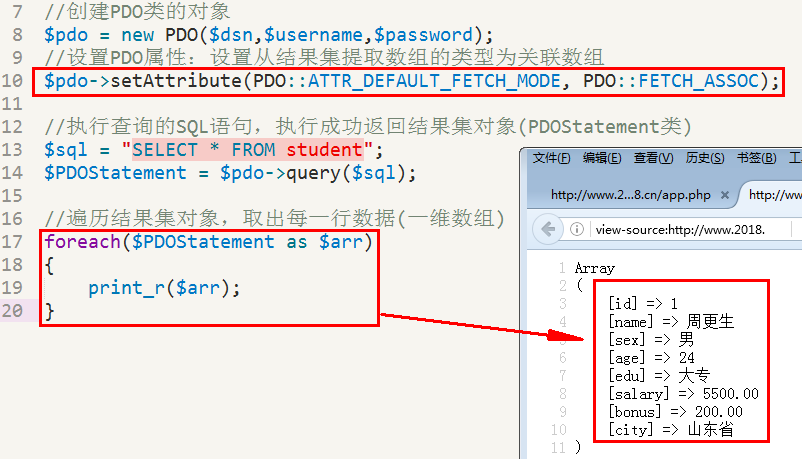
## 3、PDO::lastInsertId()方法

* 描述：返回最后插入行的ID或序列值
* 语法：**string PDO::lastInsertId ( void )**
* 返回：返回最后插入行的ID。



## 4、PDO::setAttribute()方法

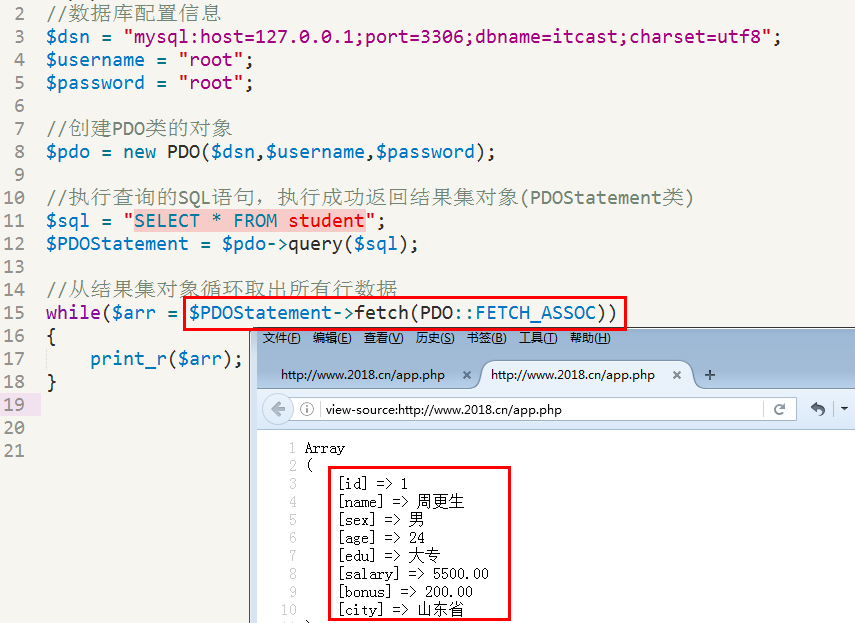
* 描述：设置数据库句柄属性
* 语法：**bool PDO::setAttribute ( int $attribute , mixed $value )**
* 提示：PDO内置了一些可用的通用属性(详细见手册)
  + PDO::ATTR\_CASE：强制列名为指定的大小写。
  + PDO::ATTR\_ERRMODE：错误报告
  + PDO::ATTR\_DEFAULT\_FETCH\_MODE： 设置默认的提取模式。
* 返回：成功时返回 TRUE， 或者在失败时返回 FALSE。



# PDOStatement对象常用方法

## 1、PDOStatement::fetch()方法

* 描述：从结果集中获取一行，并向下移动指针
* 语法：**mixed PDOStatement::fetch ([ int $fetch\_style ] )**
* 参数：$fetch\_style，控制下一行如何返回给调用者。
  + PDO::FETCH\_ASSOC，返回一个索引为结果集列名的数组
  + PDO::FETCH\_BOTH(默认)，返回一个索引为结果集列名和以0开始的列号的数组
  + PDO::FETCH\_NUM：返回一个索引为以0开始的结果集列号的数组
* 返回：此函数（方法）成功时返回的值依赖于提取类型。在所有情况下，失败都返回 FALSE 。



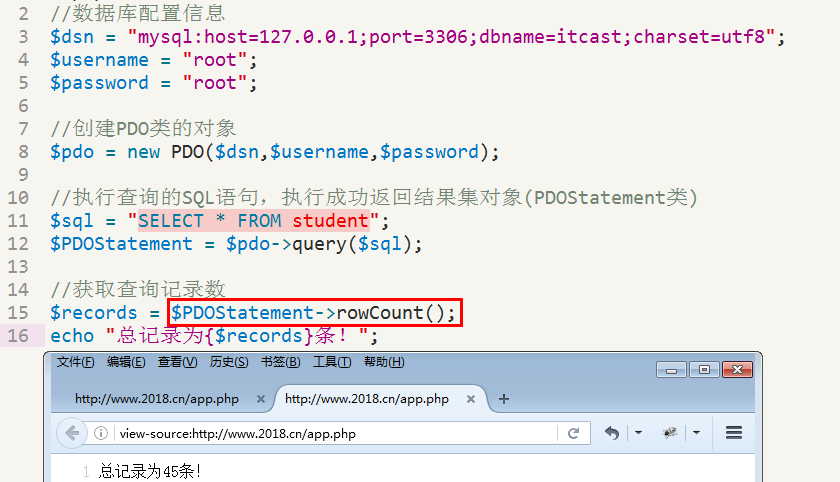
## 2、PDOStatement::fetchAll()方法

* 描述：返回一个包含结果集中所有行的数组
* 语法：**array PDOStatement::fetchAll ([ int $fetch\_style ] )**
* 参数：$fetch\_style，控制下一行如何返回给调用者。
  + PDO::FETCH\_ASSOC，返回一个索引为结果集列名的数组
  + PDO::FETCH\_BOTH(默认)，返回一个索引的数组
  + PDO::FETCH\_NUM：返回一个索引为以0开始的结果集列号的数组



## 3、PDOStatement::rowCount()方法

* 描述：返回受上一个 SQL 语句影响的行数
* 语法：**int PDOStatement::rowCount ( void )**
* 返回：返回上一个由对应的 PDOStatement 对象执行SELECT、DELETE、 INSERT、或 UPDATE 语句受影响的行数。
* 提示：要想使用该函数，必须使用 $pdo->query() 返回 PDOStatement 对象。



# PDO错误处理

## 1、PDO支持三种错误模式

* 静默模式(Silent)：错误发生后，不会主动报错，是默认的模式；
* 警告模式(Warning)：错误发生后，通过PHP标准来报告错误；
* 异常模式(Exception)：错误发生后，抛出异常，需要捕捉和处理；
* 提示：可以通过 PDO::setAttribute() 更改错误模式。

## 2、静默模式(Slient)

在静默模式下，当有错误发生时，不会显示在页面上；

此时，可以通过PDO的PDO::errorCode()和PDO::errorInfo()两个方法，来获取错误信息。



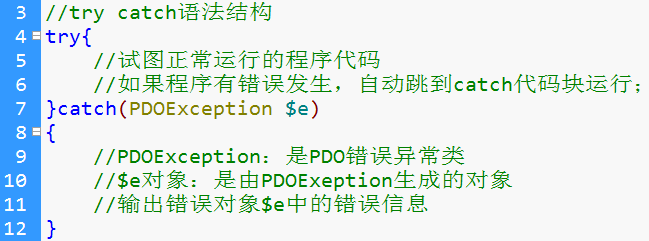
## 3、警告模式(Warning)

因为默认报错模式是静默模式(silent)，如果想报警告错误，必须使用setAttribute()方法提前设置。



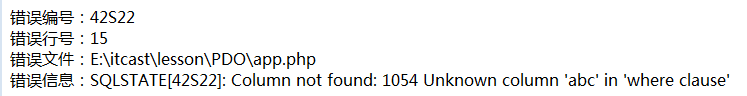
## 4、异常模式(Exception)

因为默认报错模式是静默模式(silent)，如果想报异常错误，必须使用setAttribute()方法提前设置。





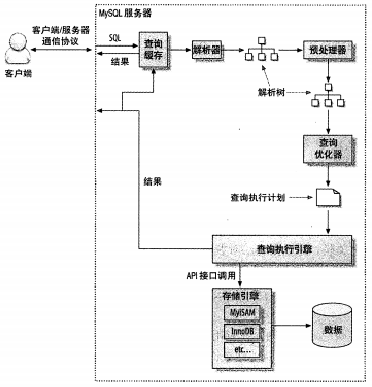




# SQL语句预处理

## 1、SQL语句执行过程

* SQL语句的执行，分成两个阶段：编译和执行。
* 如果SQL语句，是第1次执行，先编译再执行。编译过程十分复杂，耗用系统资源，相对不太安全；
* 如果SQL语句(即相同的SQL语句)，是第2次执行，直接从缓存中读取，无疑执行效率是最高的，也是比较安全的，可以有效避免SQL注入等安全问题；



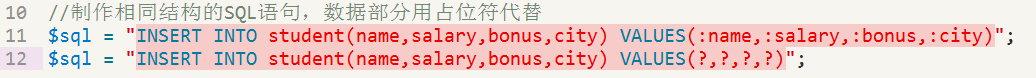
## 2、PDO的SQL语句预处理步骤

### （1）PDO完成预处理需要的步骤

* 先提取相同结构的sql部分！(将数据部分，可变的部分去掉)
* 编译这个相同的结构！将编译结果保存！
* 再将不同的数据部分进行替换！
* 执行即可！

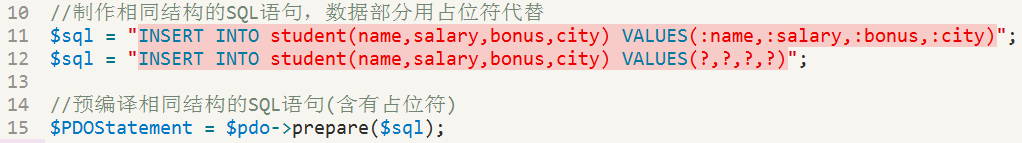
### （2）提取相同结构的SQL语句

* 在SQL语句中，使用命名参数和问号参数，来代替可变的数据。
* 使用占位符“:value”和“?”来代替可变的数据。



### （3）预编译相同结构的SQL语句

* 描述：执行预编译的SQL语句结构，并返回一个PDOStatement对象。
* 语法：**public PDOStatement PDO::prepare ( string $statement )**
* 返回：执行成功返回PDOStatement对象，失败返回FALSE。



### （4）给占位符绑定数据

* 描述：绑定一个值到预处理的 SQL 语句中的对应命名占位符或问号占位符。
* 语法：**bool PDOStatement::bindValue ( mixed $parameter , mixed $value )**
* 参数：
  + $parameter，参数标识符。对于使用命名占位符的预处理语句，应是类似 :name 形式的参数名。对于使用问号占位符的预处理语句，应是以1开始索引的参数位置。
  + $value，绑定到参数的值；
* 返回：成功时返回 TRUE， 或者在失败时返回 FALSE。



### （5）执行预处理的SQL语句

* 描述：执行一条预处理语句
* 语法：**bool PDOStatement::execute( )**
* 返回：成功时返回 TRUE， 或者在失败时返回 FALSE。



## 3、实例：PDO的SQL语句预处理

