# SQL注入

如果SQL中，存在浏览器端请求的数据（用户数据），用户通过特殊的数据形式，对SQL的行为作出影响，称之为SQL注入。（SQL injection）

"SELECT \* FROM `p34\_admin` WHERE admin\_name='$admin\_name' and admin\_pass=md5('$admin\_pass')"

正常数据：

SELECT \* FROM `p34\_admin` WHERE admin\_name='admin' and admin\_pass=md5('1234abcd')

SELECT \* FROM `p34\_admin` WHERE admin\_name='wrong' and admin\_pass=md5('abcd1234')

非正常数据：

‘ or 1#

SELECT \* FROM `p34\_admin` WHERE admin\_name='' or 1#' and admin\_pass=md5('afdadfa')

检索 条件永远为真

‘ or 1 or ‘

SELECT \* FROM `p34\_admin` WHERE admin\_name='' or 1 or '' and admin\_pass=md5('')

Tip:

不仅仅在登录时，可以被注入，任何用户数据参与执行，都可以被注入。

## 预防

业务逻辑上预防(尽可能使用白名单，将数据限制在合理的范围之内)

通过限定数据格式（比如用户名仅仅由字母数字和特定的字符组成），如果检测不满足要求，则直接不予参与SQL注入问题。

通过限定数据类型（比如，ID需要整型的数据，将用户传递请求数据强制转换成整型），进行避免。

Delete from `match` where m\_id=$\_GET[‘m\_id’]

(int) $\_GET[‘m\_id’] = 0 or 1#

特殊数据转义：

通过函数：

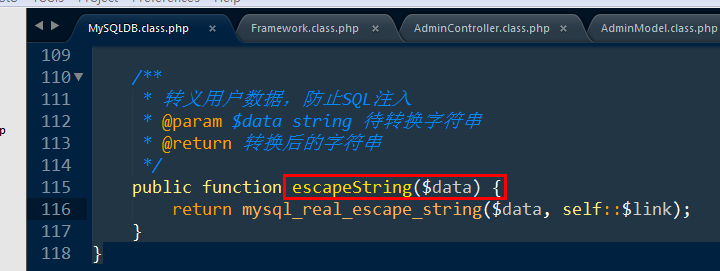
PHP字符串函数： addslashes()

MySQL提供数据转义函数：mysql\_real\_escape\_string(待字符串，连接)

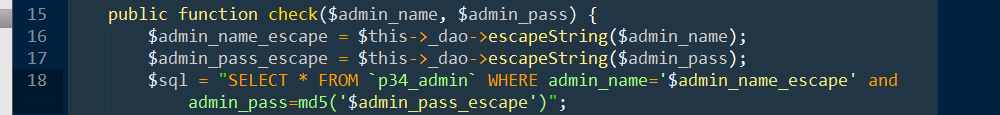


## 项目中添加防止SQL注入的代码

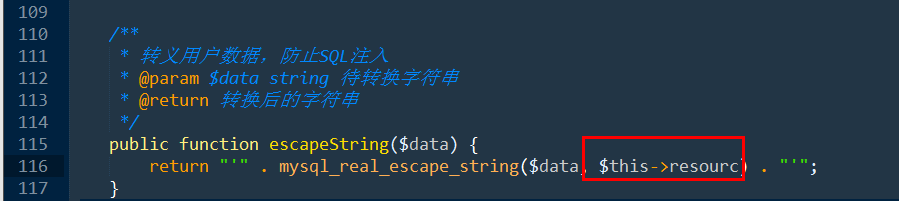
在MySQLDB中，添加一个可以完成转义的方法，在模型中，需要时调用！



项目中使用，AdminModel中，完成转换：



为了避免，整型数据可能不被SQL中增加引号，强制在转换后的数据使用引号包裹；

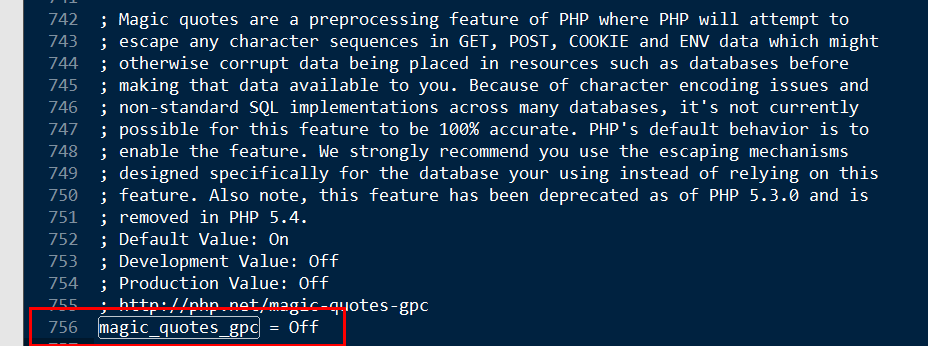


拼凑SQL的数据不需要增加引号:

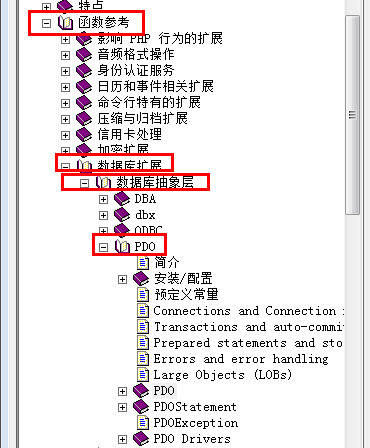


Tip:

PHP中魔术引号（magic quotes）,PHP中自动为请求数据（GET，POST）增加转义的一种防止SQL注入机制。

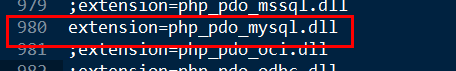


# PDO\_MySQL操作



PHP新版中推荐的基于面向对象操作数据库服务器的方式！

开启PDO\_mysql相关扩展：



Apache restart

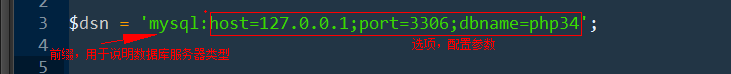
基本使用步骤

连接，认证，发送SQL，等待mysql服务器的执行结果，处理执行结果

## 实例化PDO对象

构造方法需要4个参数：

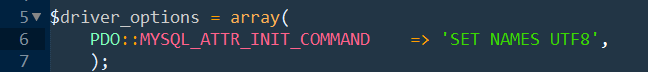
$dsn: Data Source Name数据源名称，需要的数据来源的描述



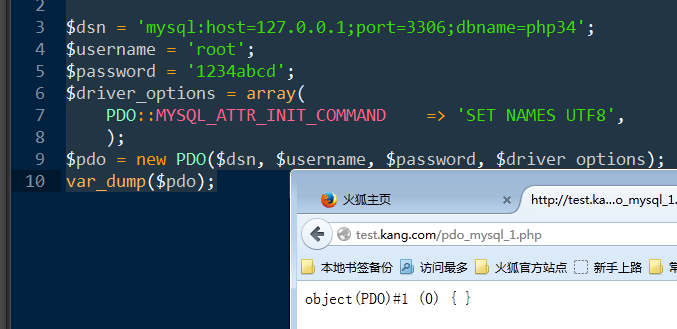
$username

$password

$driver\_options 驱动选项，对应当前数据库服务器一些特定的初始化工作。



测试：



## 发送SQL到服务器等待执行SQL的结果

Pdo对象方法query(SQL)完成执行SQL



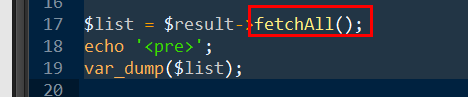
## 处理执行结果



逻辑上，返回的是结果集，被PDOStatememt类对象封装管理。

Tip：PDOStatement类，称之为PDO语句对象，SQL执行完（处理完）产生的结果对象。

利用该结果对象，获得我们需要的数据：



默认返回关联和索引的集合，可以传递参数，进行配置：



## PDO对象的常用方法

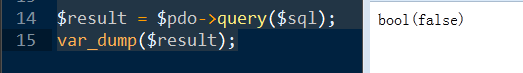
### errorInfo()

错误信息，包含错误信息的数组。

### errorCode()

错误代码

当SQL执行过程中，失败，会产生错误：





### beginTransaction();

开启事务

### rollback();

回滚事务

### commit();

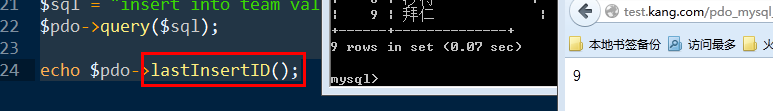
提交事务

### inTransaction();

判断是否处于事务中

### lastInsertID()

最后形成的auto\_increment字段的值。



### Exec()

又一个执行SQL的方法。

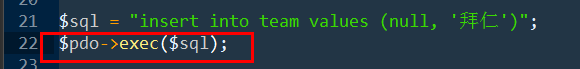
区别于query(),返回值类型不同。Exec()返回值类型为整型。(query()返回的是对象类型)。

该返回的整数，表示当前执行SQL所影响的记录数。

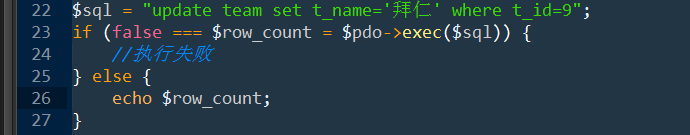
可见：

Query(), 执行查询类，show，select，desc

Exec(), 非查询类，insert，delete，update，DDL



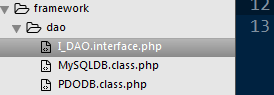
注意，影响的记录数可能为0.但是不意味着执行失败！

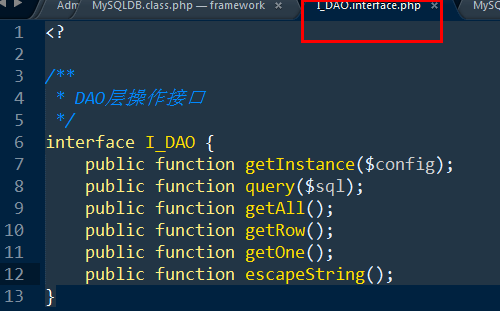


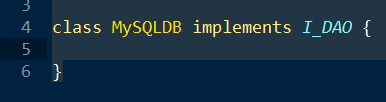
## 项目中对PDO的使用

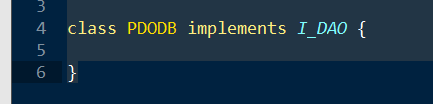
使用PDO，实现与mysqlDB这个DAO对象外部操作一致的另外一种DAO的实现，可以做到，模型切换DAO，是不需要更改模型内的任何代码（热插拔，无缝切换）

增加DAO的操作接口（I\_DAO），使所有DAO的定义（mysql扩展，PDO扩展）都实现相同的DAO操作接口（I\_DAO）









## 预处理（预编译）的执行方式

一条SQL的执行，MySQL分成两大步骤：

1. 编译；2，执行

如果：

连续执行多条结构相同的SQL：

编译的中间结果（语法树）应该是一致的。

应该将相同的结构，统一编译，每次使用不同的数据执行即可！

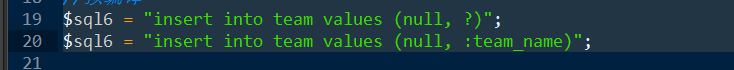
预编译：指的是预先编译SQL的结构的一种执行SQL的方法。

语法实现：

### 编译统一的结构

$PDOStatement = $pdo->prepare(SQL的结构);

SQL结构中的数据部分，可以使用问好，或者冒号标签的语法来占用：





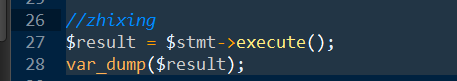
### 绑定数据到中间编译结果

$PDOStatement->bindValue()

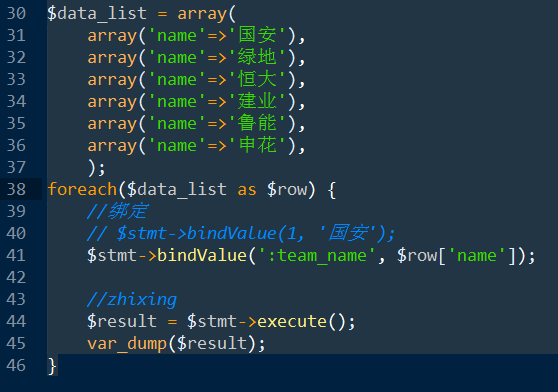


### 执行

$PDOStatement->execute();



循环执行：



预编译的优点二：

更好的防止SQL注入，是因为预编译时，不需要用户的数据参与。编译时，结构固定，导致用户数据不能影响到SQL的结构。

普通的执行方法：

$pdo->query();

$pdo->exec()

如果需要防止SQL注入，需要人为的转义用户数据，使用的方式是$pdo->quote()方法。转义并使用引号包裹。

## PDOStatement对象的常用方法

### errorInfo()

### errorCode()

使用PDOStatement->execute()方法执行失败时，错误信息由以上两个方法获得。

### fetchColumn(index=0)

允许传递参数，表示获得第一条记录的第几个字段的值。

getOne()

### Fetch()

getRow()

获取一条后，可以向后移动结果集指针。

### FetchAll()

getAll()

### rowCount();

统计处理的记录数：

影响了多少行（增删改）

结果集中存在多少行（查）。

### closeCursor();

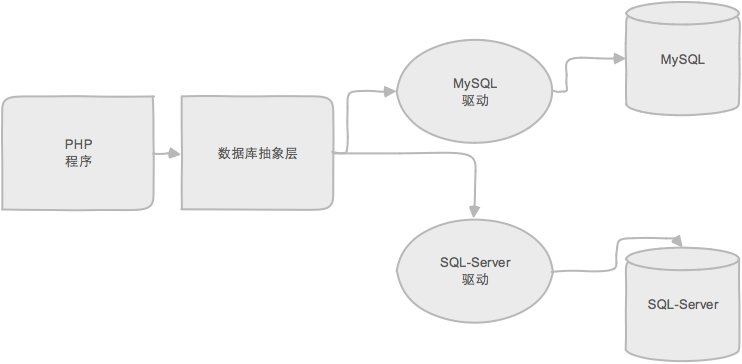
释放结果集光标。（建议fetch之后，手动释放）

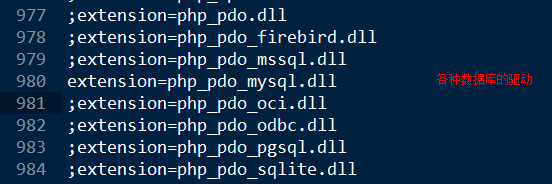
## 数据据抽象层&驱动

PDO：PHP Data Object。

PDO是数据库抽象层的一种。

将多种数据库的操作，抽象出来，使用同一种语法完成。



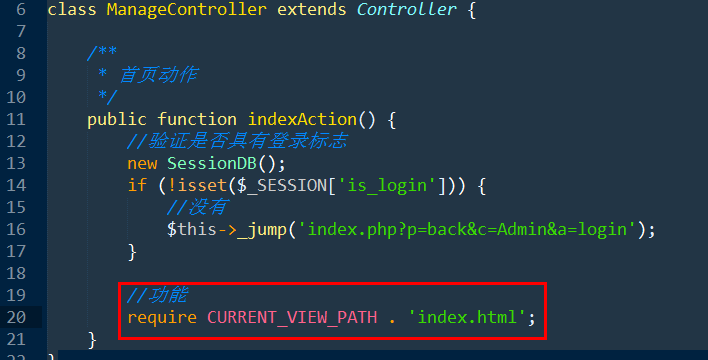


# 后台首页功能

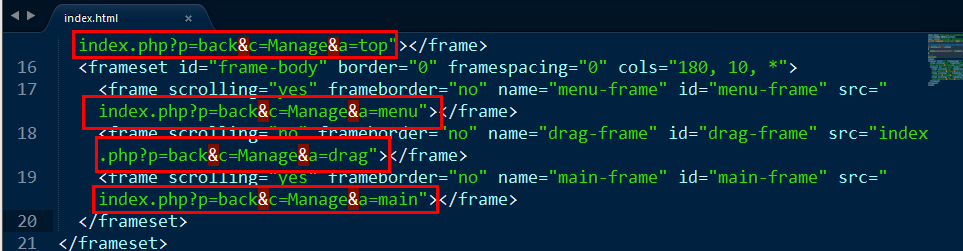
## 主体功能

框架集布局，各个框架中，都需要请求项目的某个功能动作，完成最终的展示！

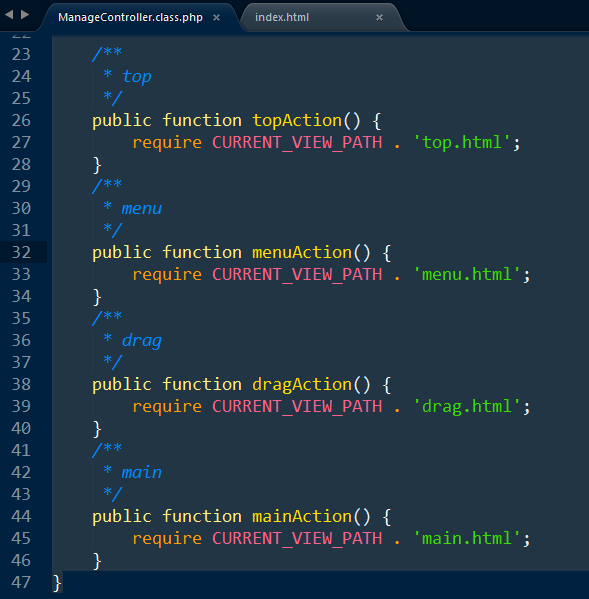
Back/ManageController->indexAction()



完成后台模板index.html



## 各个frame功能



实现对应的视图模板：

# 后台公共登录验证 – 平台控制器

后台不仅Manage->indexaction()需要登录验证，而且其他动作需要登录验证。

该操作属于当前平台后台的公共控制器功能。

增加后台的平台控制器，完成相应验证工作：



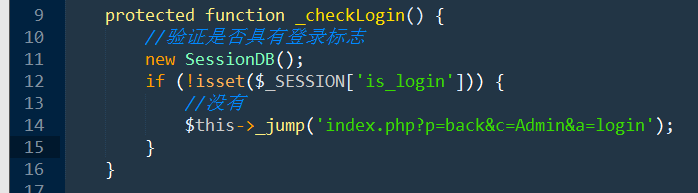
平台中其他的控制器，都继承自平台控制器：





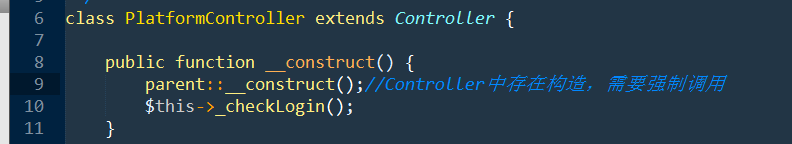
将验证登录操作，移动到PlatformController：

实现：



调用时机：

实例化后台控制器对象的时候（构造方法执行时）：



测试：



发现，已经可以验证未登录，并跳转到登录，但是登录页面没有正常加载：

Why？

请求admin的login时，也要判断是否登录，没有由跳转到admin的login，导致死循环。

可见：后台的绝大多数的动作都需要登录验证后在执行，但是存在特例：登录相关，找回密码之类。

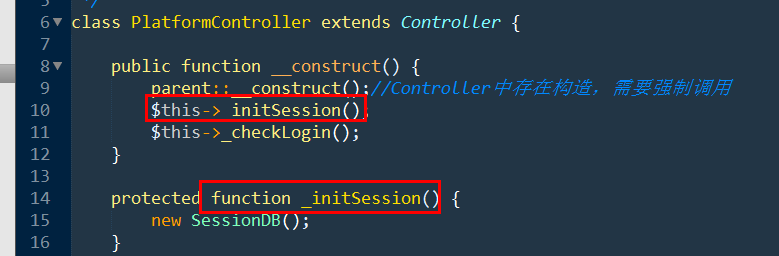
应该判断是否登录时，如果是特例动作则不判断：

如何判断特例？



由于开启session在公共的位置，后台的动作中，就不需要再次开启session了：

adminController->checkAction()

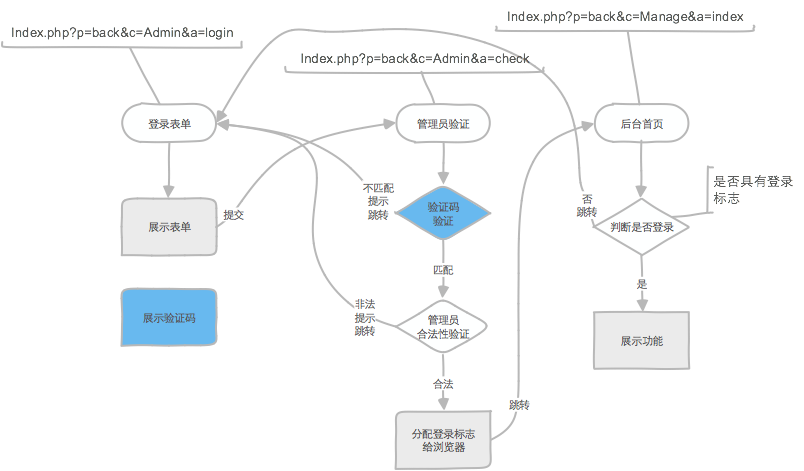


# 验证码的分析

登录：防止暴力破解

论坛：防止灌水水

展示类：被抓取。



需要技术：

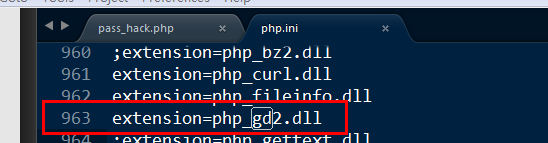
图片处理技术。

会话session技术。

# PHP图片处理技术 – GD

具体操作步骤

## 开启GD扩展



Apache restart

场景：

制作500×300的绿色图片

## 创建画布

画布，一种资源型数据。可以操作的图像资源。

创建新画布（新建）

ImageCreate(宽，高)，创建基于调色板的画布。

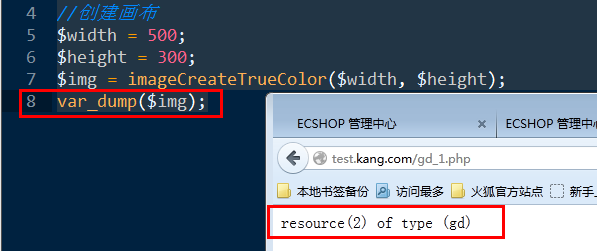
imageCreateTrueColor(宽，高);创建真彩色的画布。

基于图片创建画布（打开）

imageCreateFromJPEG(图片地址);

imageCreateFromPNG(图片地址);

imageCreateFromGIF(图片地址);



## 操作画布

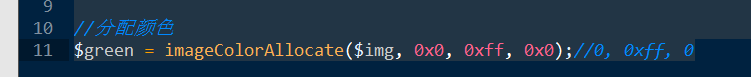
分配颜色：如果需要在画布上使用某个颜色，应该先将颜色分配到画布上。

使用函数：

颜色标识 = imageColorAllocate(画布，R，G，B);

颜色的表示方式：

RGB



填充画布：将填充点，连续并且颜色相同的点进行填充（替换）

使用函数：

imageFill(画布, 填充位置x， 填充位置Y，颜色标识)完成

位置采用坐标进行管理：

原点：0，0,画布的左上角。

向右，x轴增加，向下Y轴增加。

右下角坐标：width-1,height-1



## 输出画布

将画布中处理好的图样信息，输出出来。

典型的：

1输出到图片文件。

2直接输出。

使用函数：

imagePNG(画布[, 文件地址]):

imageJPEG();

imageGIF();

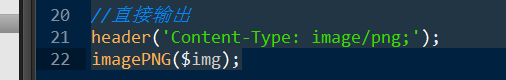
如果没有第二个参数，表示直接输出。





如果直接输出到浏览器，需要告知浏览器，响应数据的类型应该是PNG格式的图片：

使用指令Content-type



tip:一个画布可以输出多次，输出为各种格式！

## 销毁画布资源

imageDestroy();

