Определение PDO

PHP Data Objects(PDO) – легкий интерфейс для доступа к базам данных в языке PHP. Он может работать с большинством баз данных, такими как MS SQL ,Firebird, MySQL , Oracle, PostgreSQL , SQLite и другими. Но тут необходимо обратить внимание, что PDO предоставляет необходимый функционал для работы с базами данных, но для каждого типа базы данных должен быть установлен свой драйвер доступа для базы данных в виде расширения PHP.

Подключение через интерфейс PDO

Подключение довольно простое, за тем исключением, что теперь одной строкой необходимо сразу указать, к какому типу базы данных вы подключаетесь, имя хоста, а также имя базы данных.

Формат вот такой: **тип_базы_данных:host=имя_хоста;db=name**

```
try
{
        $db = new PDO('mysql:host=localhost;dbname=test','root','');
        $rows = $db->exec("CREATE TABLE `testing`(
        id INT PRIMARY KEY AUTO_INCREMENT,
        fname VARCHAR(20) NOT NULL DEFAULT '',
        email VARCHAR(50) NOT NULL DEFAULT '')");
}
catch(PDOException $e)
{
        die("Error: ".$e->getMessage());
}
```

Если же в SQL выражении вы допустили ошибку, в PDO есть специальные функции:

errorCode() – возвращает номер ошибки, и

errorInfo() – возвращает массив, в котором, как номер ошибки, так и текст описания

Запросы непосредственно можно делать двумя функциями: **exec() и query()**

Отличие их состоит в типе возвращаемого результата, exec возвращает количество затронутых в результате выполнения запроса строк, а вторая, возвращает результат запроса в объекте PDOStatement, о нем поговорим чуть ниже. *См. пример-1*

Подготовленные выражения

Подготовленные выражения очень похожи на обычные SQL запросы, но имеют некоторые преимущества. Во-первых имеют большую скорость выполнения, а во-вторых являются более надежными с точки зрения безопасности, так как все параметры передаваемые в них, автоматически экранируются от всевозможных инъекций.

Они имеют весомое преимущество в скорости, при выполнении многократных одинаковых запросов, чем если каждый раз составлять запрос заново. Также экономится траффик между приложением и базой данных.

PDO предоставляет удобные функции для работы с подготовленными выражениями. В случае, если выбранный тип базы данных не поддерживает работу с подготовленными выражениями, PDO просто будет эмулировать их работу своими методами.

И так для начала создадим подготовленное выражение, это делается функцией Prepare()

В качестве параметра она принимает SQL запрос, но в нем, вместо значений, которые необходимо менять, ставятся псевдо переменные, которые могут быть в виде знака вопроса(?), либо имени псевдо переменой, которое начинается с двоеточия (:)

```
$sth1 = $db->prepare("SELECT * FROM `testing` WHERE id=:id");
$sth2 = $db->prepare("SELECT * FROM `testing` WHERE id=?");
```

В зависимости от того, как вы определите переменную, будет зависеть ваша дальнейшая работа.

Если вы определили переменные знаком вопроса, то потом, в функцию execute передайте массив значений, в той, последовательности, в которой стоят переменные.

Если же вы обозначили переменные именами, то надо будет назначить каждой переменной значение посредством функций:

bindValue() – присваивает псевдопеременной значение

bindParam() – связывает псевдопеременную с настоящей переменной, и при изменении настоящей переменной, не нужно больше вызывать никаких дополнительных функций, можно сразу execute()

Примеры

1ый вариант:

```
$sth1 = $db->prepare("SELECT * FROM `testing` WHERE fname=?");
$sth1->execute(array('ivan'));
while($row = $sth1->fetch())
{
        print_r($row);
}
```

2ой вариант:

```
$sth3 = $db->prepare("SELECT * FROM `testing` WHERE id=:id");

for($id=1; $id < 4; $id++)
{
        $sth3->bindValue(':id',$id);
        $sth3->execute();

        while($row = $sth3->fetch())
        {
                print_r($row);

        }
}
```

Примеры

Второй вариант можно записать еще проще, связав псевдопеременную с реальной:

```
$sth3 = $db->prepare("SELECT * FROM `testing` WHERE id=:id");

$sth3->bindParam(':id',$id);

for( $id = 1; $id < 4; $id++)
{
        $sth3->execute();

        while($row = $sth3->fetch())
                print_r($row);
}
```

Тут же необходимо добавить, что очень желательно (чтобы не возникало лишних ошибок) третьим параметром указывать тип переменной. У меня лично, в случае отсутствия типа переменной, возникали ошибки в операторе WHERE, так как он считал переменную текстом, а не числом.

$sth3->bindParam(':id',$id, PDO::PARAM_INT);

$sth3->bindParam(':id',$id, PDO::PARAM_STR);

Еще одним из очень приятных плюсов использования таких подготовленных выражений, это экранирование переменных. Перед подстановкой в процедуру все переменные экранируются и никакие SQL инъекции не страшны.

Транзакции

Транзакция – это совокупность запросов базу данных, которые должны быть обязательно выполнены все. Если какой-либо запрос не выполнен или выполнен с ошибкой, то транзакция отменяется и изменений данных в базе не происходит.

Это нужно, чтобы гарантировать сохранение целостности данных при нескольких запросах. например при переводе денежных средств со счета на счет.

Чтобы выполнить транзакцию в PDO необходимо перейти в режим ручного подтверждения запросов.

Чтобы выключить режим автоподтверждения, выполняем команду:

$db->beginTransaction();

После этого выполняем столько запросов к базе данных сколько необходимо сделать в этой транзакции. И только после того как все запросы будут выполнены, Вы можете подтвердить транзакцию функцией **$db->commit();**
или отменить транзакцию **$db->rollback();** Тут еще следует заметить, что не все типы таблиц поддерживают транзакции, требуется ипользовать таблицу InnoDb вместо стандартной MyISAM. *см. Пример-2*

Проблема при разработке ПО

Многие разработчики очень сильно усложняют код и потом его дорого обслуживать и развивать или приходится делать глобальный рефакторинг, что довольно не выгодно для бизнеса.

Основная цель — это выполнить проект в сроки и в дальнейшем развивать его вкладываясь в запланированный бюджет.

Один из важных критериев качества кода — это его простота. Но как измерять простоту? Один из вариантов — это рассчитать кол-во элементов системы. Чем меньше элементов тем система проще.

Что такое архитектура программы

Архитектура - это базовая организация системы, воплощенная в ее компонентах, их отношениях между собой и с окружением, а также принципы, определяющие проектирование и развитие системы.

Другими словами

- **Компоненты кода *(«на какие части разбить»)***
- **Структура кода *(«где, что будет лежать»)***
- **Отношения между компонентами**

Что такое хороший код

1. **Решающий поставленную задачу**
2. **Делающий это наиболее очевидным образом**
3. **Экономящий ресурсы**
4. **Расширяемый**
5. **Аккуратный**

Модель MVC

MVC(Model-view-controller, «*Модель-представление-поведение*», «*Модель-представление-контроллер*») — это шаблон проектирования приложений, при котором управляющая логика поделена на три отдельных компонента таким образом, что модифицирование одного из них дает минимальное влияние на остальные.

Шаблон MVC хорошо применять при создании сложных проектов, где необходимо отделить работу php программиста (или разделить группу программистов на отделы), дизайнера, верстальщика, и т.д.

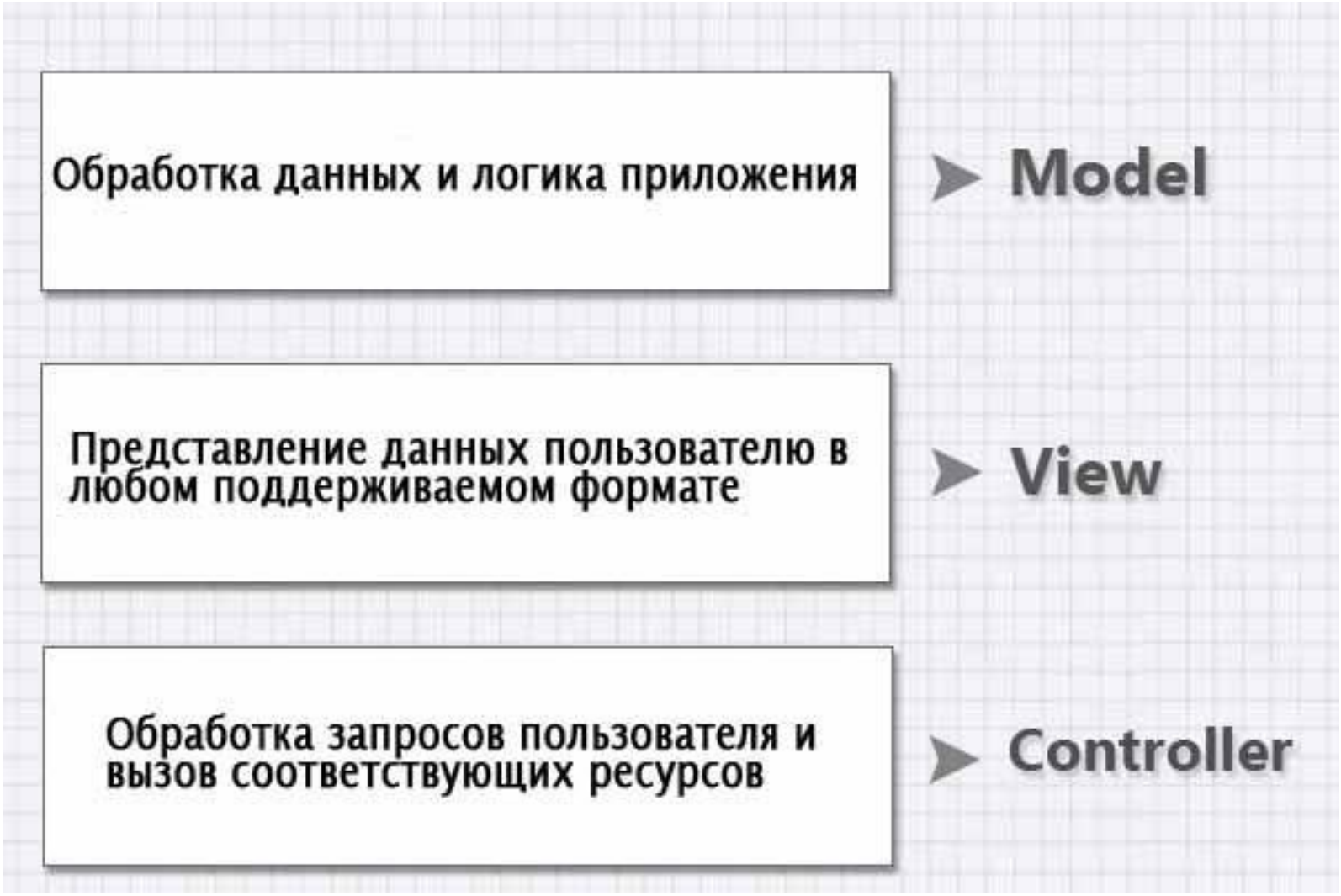
Шаблон MVC разделяет представление, данные, и обработку действий пользователя на три отдельных компонента:

MVC Модель (Model). Модель предоставляет данные (обычно для View), а также реагирует на запросы (обычно от контроллера), изменяя своё состояние.

MVC Представление (View). Отвечает за отображение информации (пользовательский интерфейс).

MVC Поведение (Controller). Интерпретирует данные, введённые пользователем, и информирует модель и представление о необходимости соответствующей реакции.

Модель MVC


Обработка данных и логика приложения
Model
Представление данных пользователю в любом поддерживаемом формате
View
Обработка запросов пользователя и вызов соответствующих ресурсов
Controller

Благодарю за внимание!