**Проектирование информационной системы.**

При разработке информационных систем часто используют шаблоны проектирования, которые представляют собой формализованное описание часто встречающейся задачи проектирования, удачное решение данной задачи, а также рекомендации по применению этого решения в различных ситуациях. Кроме того, шаблон проектирования обязательно имеет общеупотребимое наименование. Правильно сформулированный шаблон проектирования позволяет, отыскав однажды удачное решение, пользоваться им снова и снова.

Модель системы, построенная в терминах шаблонов проектирования, фактически является структурированным выделением тех элементов и связей, которые значимы при решении поставленной задачи. Помимо этого, модель, построенная с использованием шаблонов проектирования, более проста и наглядна в изучении, чем стандартная модель.

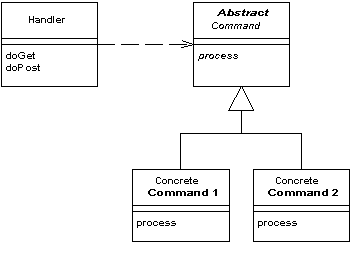
Применение шаблонов проектирования повышает устойчивость системы к изменению требований и упрощает неизбежную последующую доработку системы.

Ниже будут рассмотрены шаблоны проектирования, которые были использованы при разработке системы.

**Шаблон проектирования** **Front Controller (Контроллер входа / Единая точка входа).**

В сложных веб-сайтах есть много одинаковых действий, которые надо производить во время обработки запросов. Это, например, контроль безопасности, многоязычность и настройка интерфейса пользователя. Когда поведение входного контроллера разбросано между несколькими объектами, дублируется большое количество кода. Помимо прочего возникают сложности смены поведения в реальном времени.

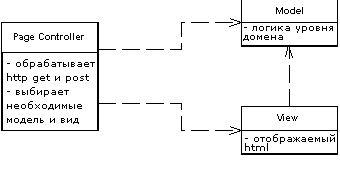
Паттерн Front Controller объединяет всю обработку запросов, пропуская запросы через единственный объект-обработчик. Этот объект содержит общую логику поведения, которая может быть изменена в реальном времени при помощи декораторов. После обработки запроса контроллер обращается к конкретному объекту для отработки конкретного поведения.



**Шаблон проектирования** **Page Controller (Контроллер страницы).**

Когда происходит запрос к статической HTML-странице, веб-серверу передаётся имя и путь к хранящемуся на нём HTML-документу. Главная идея здесь в том, что каждая страница на веб-сайте является отдельным документом, хранящимся на сервере. В случае с динамическими страницами всё гораздо сложнее, так как сложнее связь между введённым адресом и отображённой страницей. Тем не менее, подход, когда один путь соответствует одному файлу, который обрабатывает запрос достаточно очевиден и прост для понимания.

В результате контроллер страницы (Page Controller) - паттерн, в котором один контроллер отвечает за отображение одной логической страницы. Это может быть как отдельная страница, хранящаяся на веб-сервере, так и отдельный объект, который отвечает за страницу.



**Шаблон проектирования MVC - Model View Controller (Модель-Вид-Контроллер).**

Этот шаблон разделяет работу веб-приложения на три отдельные функциональные роли: модель данных (model), пользовательский интерфейс (view) и управляющую логику (controller). Таким образом, изменения, вносимые в один из компонентов, оказывают минимально возможное воздействие на другие компоненты.

В данном паттерне модель не зависит от представления или управляющей логики, что делает возможным проектирование модели как независимого компонента и, например, создавать несколько представлений для одной модели.

Впервые этот шаблон был применён в фреймворке, разрабатываемом для языка Smalltalk в конце 1970-х годов. С этого момента он играет основополагающую роль в большинстве фреймфорков с пользовательским интерфейсом. Он в корне изменил взгляд на проектирование приложений.

Большинство фреймворков для веб-программирования сейчас в основе своей содержат именно MVC.

**Шаблон проектирования Singleton(Одиночка).**

При разработке больших проектов, часто возникает необходимость обращаться из одного модуля программы к объектам, существующим в другом модуле.

Такие объекты, как правило, существуют в единичных экземплярах, поэтому наиболее распространенной практикой является создание глобальных объектов данного типа и ссылка на них из других модулей программы. Но поскольку объект создается статически, отсутствует возможность управлять процессом его создания. То есть, объект создается автоматически, до момента его фактического применения в программе. Это может приводить к некоторым неприятным последствиям.

Выходом является использование паттерна проектирования Singleton. Его особенностью является то, что он гарантирует существование объекта в единственном экземпляре, а самое главное, то, что он создается в тот момент, когда это требуется клиенту. Последующие попытки конструирования объекта приводят лишь к возвращению клиенту ссылки на уже существующий объект, но не к созданию нового.

Реализация шаблона проектирования Singleton на языке PHP:

*<?php*

*class Singleton {*

*protected static $instance;*

*private function \_\_construct() { /\* ... \*/ }*

*public static function getInstance() {*

*if ( is\_null(self::$instance) ) {*

*self::$instance = new self;*

*}*

*return self::$instance;*

*}*

*public function doAction() { /\* ... \*/ }*

*}*

*Singleton::getInstance()->doAction();*

*?>*

**Основные модули системы.**

Работы системы разделена на несколько основных модулей, связанных между собой. Основные модули:

* Формирование и вывод страницы (класс PAGE).
* Работа с базой данных (классы DB, SqlData).
* Работа с каталогами (классы Catalog, Rubric, Item).

Более подробно эти модули рассмотрены ниже.

**Формирование и вывод html страницы.**

Взаимодействие пользователя с сайтом осуществляется через единую точку входа – файл index.php, находящийся в корне сайта. В нем подключаются все модули системы, загружается шаблон сайта и передается управление шаблону.

*require\_once($\_SERVER['DOCUMENT\_ROOT']."/core/core.php");*

*$name = getVar('site\_template');*

*if(!$name) $name='default';*

*include $\_SERVER['DOCUMENT\_ROOT']."/templates/{$name}/index.php";*

Шаблон сайта, получив управление, создает объект класса PAGE. Далее происходит загрузка шаблона страницы и подключение контролеров.

*$page = PAGE::get();*

*$page->assign('rubrics\_list', 'inc/common/rubrics\_list.php');*

*$page->assign('menu', 'inc/common/menu.php');*

*$type = $\_GET['type'];*

*if ($type == 'display') $page->assign('content', 'inc/common/display.php');*

*else if ($type == 'catalog') $page->assign('content', 'inc/common/catalog.php');*

Когда скрипт заканчивает работу, вызывается деструктор класса PAGE и функция PAGE::show(). Во время работы этой функции происходит:

* Поиск готовой страницы в кэше.
* Если страница не найдена, то подключение контролеров и сохранение страницы в кэш.
* Выполнение действий контролеров.
* Проставление спец-меток page\_title, page\_description, page\_keywords.
* Вывод готовой страницы в браузер пользователя.

**Работа с базой данных.**

Для работы с базой данных MySQL был написан класс DB на основе шаблона проектирования Singleton. При создании экземпляра класса загружаются настройки подключения к базе данных с создается соединение с базой.

*private function \_\_construct()*

*{*

*include ($\_SERVER['DOCUMENT\_ROOT'].'/admin/mysql.php');*

*$this->connection = mysql\_connect($mysql\_host, $mysql\_login, $mysql\_psswd) or die(mysql\_error);*

*mysql\_select\_db($mysql\_dbname);*

*mysql\_query("SET NAMES 'cp1251'");*

*}*

Запросы к базе выполняет функция DB::sql.

*public function sql($query)*

*{*

*$result = mysql\_query($query, $this->connection);*

*if (!$result)*

*{*

*echo $query;*

*echo mysql\_error();*

*die();*

*}*

*return $result;*

*}*

Для более короткой записи объявлена глобальная функция sql.

*function sql($query)*

*{*

*return DB::get()->sql($query);*

*}*

Пример запроса к базе данных:

*$result = sql(‘SELECT \* FROM catalogs’);*

Использование шаблона проектирования Singleton позволяет обращаться к базе из любого места приложения и гарантирует создание только одного подключения к базе. Благодаря единой функции работы с базой, можно вести историю запросов для последующей оптимизации обращений к базе.

Работу с записями базы данных реализует класс SqlData. Для получения данных об объекте из базы необходимо знать таблицу, ключевое поле и значение этого поля для данного объекта. Эти параметры передаются в конструктор класса SqlData.

*public function \_\_construct($table, $selector, $selector\_value)*

Для получения данных об объекте необходимо вызвать функцию SqlData:: getData. После выполнения запроса к базе данные кэшируются и при последующих вызовах функции берутся из кэша.

*public function getData($field='')*

*{*

*if ($field == $this->\_selector) return $this->\_selector\_value;*

*else*

*{*

*if (empty($this->\_data))*

*{*

*$query = "SELECT \* FROM {$this->\_table} WHERE {$this->\_selector} = '{$this->\_selector\_value}'";*

*$this->\_data = mysql\_fetch\_assoc(sql($query));*

*}*

*if (empty($field)) return $this->\_data;*

*else if (array\_key\_exists($field, $this->\_data)) return $this->\_data[$field];*

*else return null;*

*}*

*}*

Для изменения данных объекта используются функции:

* SqlData:: setData($data) – выставление объекту заранее известных данных.
* SqlData:: addData($data) – добавление новых данных к существующим.
* SqlData:: updateData($data) – изменение данных с сохранение в базу данных.
* SqlData:: clear() – очистка кэша.

Пример получения данных об объекте:

*$name = $object->getData(‘name’);*

**Работа с каталогами.**

Класс Catalog реализует интерфейс работы с каталогом. Для создания экземпляра класса необходимо задать имя каталога.

*function \_\_construct($name)*

*{*

*$this->\_name = $name;*

*$this->\_data = new SqlData('catalogs', 'name', $name);*

*}*

Для получения доступа к каталогу существует глобальная функция getCatalog(), которая возвращает экземпляр класса Catalog.

*function getCatalog($name='')*

*{*

*if ($name=='')*

*{*

*if (($\_GET['type']=='catalog')||($\_SERVER['PHP\_SELF'] == '/admin/index.php')) $name=$\_GET['name'];*

*else return null;*

*}*

*$handle = new Catalog($name);*

*return $handle;*

*}*

Для доступа к рубрикам и позициям каталога используются функции Catalog::getRubric() и Catalog::getItem(). Они возвращают экзумпляры класса Rubric и Item соответственно. Классов Catalog, Rubric и Item содержат функцию getUrl() для получения ссылки на страницу объекта.

Основные функции при работе с каталогами:

* Catalog:: showNavChain – вывод навигационной цепочки.
* Rubric:: getParent – возвращает рубрику-родитель.
* Rubric:: getChain - возвращает массив рубрик от корневой до данной рубрики.
* Rubric:: getSubRubrics – возвращает массив подрубрик.
* Rubric:: getItems – возвращает массив элементов рубрики.
* Rubric:: addSubRubric – добавление подрубрики к данной рубрике.
* Rubric:: addItem – добавление элемента к данной рубрике.
* Item:: getRubric – возвращает рубрику, в которой находится элемент.