**Документация программного комплекса PhpZero**

# Главная

## Просто

CMF “PhpZero” относительно проста по своему устройству и не требовательна к программной среде и ресурсам. Это позволяет быстро освоиться и начать ее использовать.

## Быстро

CMF “PhpZero” обладает всем необходимым набором инструментария для разработки конкретного пользовательского функционала. Скорость работы системы высока за счет лаконичного построения и взаимодействия компонентов между собой. А также их простоты.

## Универсально

В системе реализован необходимый минимум для самой работы системы.

Простая и в тоже время гибкая и функциональная система прав и пользовательских условий.

Все остальное это инструментарий (с возможность переопределения и расширения) универсально подходящий для создания практически любого проекта в сети интернет.

# О проекте

Рады видеть Вас на нашем сайте.

Сайт посвящен программному комплексу (далее система) CMF “PhpZero”

CMF “PhpZero” – это каркас для проектирования систем управления контентом.

На его основе можно создавать системы управления содержимым (CMS), а также другие веб-приложения и сервисы как-то: сайты визитки, интернет магазины, информационные сайты, веб ориентированные ERP и/или CPM системы т. д.

## Преимущества

Если основная задача универсальных CMS — простота создания сайта без вмешательства программиста,

CMF “PhpZero” — это конструктор CMS (в том числе узкоспециализированных) для программиста. Благодаря такому подходу сайт, разрабатываемый с помощью CMF, по сравнению с сайтом на базе CMS, может иметь более простую и безопасную в работе административную панель (в которой отсутствуют функции настройки сайта под любые функции) и быть менее требовательным к ресурсам системы (каждый разрабатываемый модуль реализует именно те функции, которые необходимы в работе сайта).

При наличии грамотно спроектированной CMF и готовых базовых модулей, программисту остаётся только доработка (по необходимости отсутствующих функций) и сборка этих модулей в единое пространство с оформлением вывода в соответствии с пожеланиями клиента, то есть разработка сайта на основе CMF вполне сравнима в большинстве случаев по трудоёмкости с разработкой сайта на универсальной CMS. Но c очевидными преимуществами в производительности, управляемости и что самое главное дальнейшей масштабируемости.

С другой стороны при работе с голыми фреймворками пользователь-программист может столкнуться с программными ошибками. Не будучи экспертом в архитектуре сложного фреймворка, он может испытывать затруднения с их локализацией, не может быть уверен, сделал ли он ошибку сам, или столкнулся с ошибкой в фреймворке. Глубокие знания архитектуры фреймворка могут понадобиться и при его тонкой настройке, если программист желает получить максимальную производительность.

## Архитектура

CMF “PhpZero” на уровне файловой структуры реализована паттерном MVC

Веб-фреймворк обеспечивает прозрачную интеграцию всех трёх слоёв MVC архитектуры.

Фреймворк скрывает от программиста детали подключения к базе данных и формирования веб страниц с помощью шаблонов-представлений, позволяя ему сконцентрироваться на реализации бизнес-логики. Процесс создания приложения с использованием MVC-фреймворка заключается в написании классов контроллеров, моделей и представлений, каждый из которых является наследником базовых классов для компонентов каждого слоя. Что в свою ускоряет разработку благодаря использованию наследуемого функционала. CMF “PhpZero” обеспечивает полуавтоматическое создание прототипа веб-приложения, выполняющего основные (CRUD) операции с определенным путём скаффолдинга (метапрогарммирвания).

Любой компонент системы можно переопределить частично либо полностью!

Классы слоя Model могут быть реализованы по разным технологиям.

Одна из основных это технология ORM (Объектно-реляционное отображение)

ORM — технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных».

### Модели

Взаимодействие модели с различными компонентами системы реализовано с помощью паттерна Композиция с делегированием. В основе создания моделей лежит паттерн фабрика и фабричный метод. Что позволяет переопределять и гибко настраивать последние.

### Контролеры

В основе (базовом контролере) работы любого контролера лежит паттерн Цепочка команд (чанки).

Что позволяет разбивать логические участки кода на чанки (куски) и гибко управлять их выполнением. Также это позволяет избежать дублирования кода при полиморфизме (необходимости продублировать большую часть кода из-за не большего изменения в логике наследуемого контроллера).

Оптимизация и ускорения работы при таком подходе на лицо.

### Виды или представления

В систему встроен простой и в тоже время универсальный шаблонизатор.

Который позволяет использовать обычный php синтаксис, удобно обрамленный в сигнатуру верстки.

Одна из его сильных сторон реализация директивы плагинов. Что позволяет иерархически (рекурсивно) подключать их друг в друга и тем самым использовать один и тот же плагин многократно в любом месте, где это нужно без дублирования кода. Плагины (plugin|bar) это те же контроллеры, их упрощенная разновидность. Что в свою очередь уменьшает вероятность ошибок и сокращает время разработки. Для многократного подключения простых блоков верстки реализована директива подключений (inc|include).

Также реализована директива переводов (translation|lang)

Работа и взаимодействие вышеописанной архитектуры и его частей в целом централизованно производится и управляется компонентом приложения (Zero\_App) .

Этот компонент инициализирует систему и среду приложения, запускает на выполнение контроллеры,

формирует вывод результата и отдает его браузеру. Также отслеживает и фиксирует всю производимую работу и возникающие ошибки.

# Быстрый старт (Quick start)

В данном разделе будет описано, как максимально быстро установить систему и начать с ней работать.

## Установка системы

Предполагается, что у Вас уже имеется настроенный UNIX хостинг (БД Mysql, Php, Apache или Nginx).

Работающий на нем домен.

1. Скачиваем CMF “PhpZero” в виде zip архива [здесь](https://github.com/kshamiev/phpzero) и распаковываем. Или клонируем в корень сайта. В одноименную папку “phpzero”. К примеру: /home/domains/www.domain.com/www/phpzero
2. Запускаем процесс инсталляции, пройдя по ссылке: [http://www.domain.com/phpzero/setup](http://you-domain.com/phpzero/setup)

Следуем инструкциям установки

1. Все. Заходим на наш сайт.

### Базовый интерфейс системы

После установки системы управляющая часть сайта далее CP будет доступна по адресу (по умолчанию)

[http://www.domain.com/zero](http://you-domain.com/zero)

В системе изначально заведен системный пользователь. Логин и пароль “dev”

Системные разделы:

Система

Разделы Управление разделами (страницами) сайта (роутинг), контентом разделов.

Файловый менеджер Файловый менеджер. Редактирование текстовых файлов.

Обслуживание Обслуживающий инструментарий системы.

Контроллеры Контроллеры сайта (синонимы шаблонам).

Опции Опции проекта (Различные параметры)

Пользователи

Пользователи Работа с пользователями

Группы Работа с группами и правами доступа

Сайт

Контент Контент блоки на страницах сайта (реализуются через плагины)

*Примечание:*

Самую актуальную версию CMF “PhpZero” можно скачать отсюда (ветка master):

<https://github.com/kshamiev/phpzero>

## Принятая нумерация версий

X.X.X или R.A.B

R (Release) - Первая цифра обозначает номер релиза. Итерация этого порядка привязывается к завершению этапа разработок и выкатыванию. Как правило, это происходит при разработке каких-либо новых функциональных возможностей (модулей) или фундаментальных изменений самой системы. Или когда их количество и качество становится слишком большим.

A (Advanced) – Вторая цифра обозначает номер дополнения к релизу. Итерация этого порядка происходит при разработке нового или доработке и расширении ранее реализованного функционала.

Эта цифра обнуляется во время итерации первой цифры R (при выкатывании нового релиза).

B (Bugfix) – Третья цифра обозначает номер исправленной ошибки в работающем проекте. Итерация этого порядка происходит при исправлении выявленных ошибок в уже работающем проекте.

Эта цифра обнуляется во время итерации первой цифры R (при выкатывании нового релиза).

# Руководство (Guide, Manual)

## Терминология

**Определение, Утверждение:**

Таким заголовком обозначаются различного рода утверждения и определения, используемые в системе.

*Примечание, Совет:*

Таким заголовком обозначаются дополнительные сведения, раскрывающие тонкости содержания раздела и/или рекомендации

*Важно:*

Таким заголовком обозначаются важные предупреждения, в том числе и критические. Не соблюдение которых, может привести к потере данных или работоспособности.

Инструкции, конструкции

Таким текстом пишутся конкретные конструкции языка, системы и примеры программного кода.

## Технические директивы и стиль кодирования

Информация написанная в данном разделе и его подразделах носит рекомендательный характер.

И строго справедливо только для ядра системы.

Стиль кодирования для IDE PhpStorm (/phpzero/data/phphstorm\_code\_style.jar).

Имена переменных, констант, функций, классов, методов в подавляющем большинстве состоять из 3 словоформ. Наличие каждого из них и их порядок опциональны и зависят от контекста.

1. Группирующий префикс, обозначающий принадлежность, к какому либо значимому объединению, либо действию. Например: имя модуля, метод действия, чанк, системная переменная, операции получения, присвоения, загрузки.
2. Имя нарицательное. Собственно с чем работаем. Например: новость, группа, товар, раздел.
3. Суффикс, определяющий характер операции. Например: Добавление, удаление, калькуляция

$shop\_order\_count;

$order\_count;

$count;

Переменные, содержащие в себе публичные данные - используемые во многих местах пишутся с заглавной буквы.

Переменные, содержащие в себе приватные данные - используемые инкапсулировано в небольшом участке кода, алгоритмичные пишутся со строчной буквы.

Имена всех переменных могут использовать верблюжью нотацию либо символ нижнего подчеркивания.

*Рекомендуется верблюжья нотация.*

Имена констант пишутся в верхнем регистре и используют символ нижнего подчеркивания.

Все имена любых переменных и констант должны быть правильными словами языка. Сугубо личные ассоциации в названиях, не понятные для большинства категорически не приветствуются.

### Определенные имена переменных

$Object, $Model Абстрактный объект (новость, пользователь, товар).

$View Представление

$source, source\_name Имя источника (таблицы) хранящей целевые объекты

Пользовательские функции начинаются с префикса app\_ и располагаются в app/function.php

Системные функции начинаются с префикса zero\_ и располагаются в phpzero/function.php

Пользовательские константы начинаются с префикса APP\_ и располагаются в конфигурационном файле.

Системные константы начинаются с префикса ZERO\_

## Файловая структура

В основе реализации файлового представления системы лежит паттерн MVC.

Имена таблиц, их моделей, контроллеров, шаблонов представления, хранилища бинарных данных и кеша как правило в корне совпадают (но не аксиома). Удобство реализации и простота восприятия и понимания.

Структура:

/app

/assets Хранилище оформления шаблонов представления (img/css/js)

/i18n Переводы

/class Пространство модулей (модели, контроллеры)

/modulName Модуль (уровень компонентов и моделей)

/ModelName.php Модель

/ComponentName.php Компонент (характерно для системы “Zero”)

/Admin Административные контроллеры (web)

/Page Публичные (лицевые) контроллеры (web)

/Api Контроллеры реализующие API

/Cron|Console Консольные контроллеры (crontab)

/Bar|Plugin Плагины (функциональные контроллеры) (web)

…

/view Представления и виджеты (уровень макетов или главных шаблонов)

/LayoutName.html Шаблон макета или главный шаблон

/modulName Модуль (уровень макетов или главных шаблонов)

/LayoutName.html Шаблон макета или главный шаблон

/ViewFoder Шаблоны модуля (контроллеров) во вложенных папках

/ViewName.html Шаблон конкретного контроллера

…

Могут быть и другие папки с нужным по смыслу и функционалу содержанием

config.php – Общая конфигурация для всего приложения (может лежать в корне сайта).

function.php – Пользовательские функции приложения.

routeApi.php - Конфигурация роутинга Api контроллеров (в случае работы без БД).

routeWeb.php - Конфигурация роутинга Web контроллеров (в случае работы без БД).

routeConsole.php - Конфигурация роутинга Cron контроллеров (в случае работы без БД).

Модули (область или пространство модулей) рекомендуется формировать на основе бизнес задач либо функционального назначения.

Примеры функциональных модулей:

app/Config какие либо конфигурации, относящиеся ко всему проекту в целом и в частностях.

app/Request расширения компонента Request для реализации запросов к внешним ресурсам и сервисам. Со своими уникальными реализациями.

Примеры бизнес модулей:

app/Client модуль по работе с клиентом

app/Shop модуль реализующий продажи

app/Catalog модуль каталога продукции

app/Content модуль информационного контента на сайте

Модули можно делать вложенными. То есть модуль является собирательным образом для вложенных в него родственных модулей по функциям или бизнес задачам. (Таким образом построен модуль ядра системы)

app/Shop модуль реализующий продажи

/Domains …(Api, Admin, Page…)

/Dedicated …(Api, Admin, Page…)

/Ssl …(Api, Admin, Page…)

/Cloud …(Api, Admin, Page…)

/Vps …(Api, Admin, Page…)

Старая структура:

/application

Область модулей приложения

/modulName Модуль

Стандартный модуль содержит в себе следующие папки:

/assets Хранилище оформления шаблонов представления (img/css/js)

/class Модели, компоненты и контроллеры.

/ModelName.php Модель

/ComponentName.php Компонент

(характерно для базового модуля приложения -zero)

/ControllerFoder Контроллеры модуля во вложенных папках

/ControllerName.php

…

/i18n – Переводы

/view – Представления модуля

В модуле могут быть и другие папки с нужным по смыслу и функционалу содержанием

config.php – Общая конфигурация для всего приложения (может лежать в корне сайта).

function.php – Пользовательские функции приложения.

routeApi.php - Конфигурация роутинга Api контроллеров (в случае работы без БД).

routeWeb.php - Конфигурация роутинга Web контроллеров (в случае работы без БД).

routeConsole.php - Конфигурация роутинга Cron контроллеров (в случае работы без БД).

/assets

Хранилище оформления и клиентского интерактива сайта (img/css/js).

/exchange

Служебная папка для импорта и экспорта данных в систему и из нее.

(папка может располагаться на уровень выше)

/upload

Хранилище загружаемых бинарных данных пользователем.

/phpzero

Система или ядро. Его структура аналогична /app

/console.php Менеджер консольных задач и их выполнение (точка входа для задач через crontab)

/function.php Системные функции приложения.

.htaccess – Дополнительные настройки для сервера apache

index.php – Точка входа в приложение (web, api).

config.php – Общая конфигурация для всего приложения.

За пределами DOCUMENT\_ROOT

/cache

Хранилище кешированных данных. Содержимое не важно, и может быть удалено в любой момент.

/logs

Профилированные статистические данные работы приложений.

Все программные компоненты представлены классами (ОПП).

Их подключение происходит автоматически при первом обращении к ним через автозагрузчик (ZEND).

Алгоритм поиска и подключений классов на примере Zero\_Users\_Profile:

1. В области приложений (/application). По следующей маске: zero/class/Users/Profile.php
2. В области приложений (/app). По следующей маске: class/Zero/Users/Profile.php
3. В области ядра (/phpzero). По следующей маске: class/Zero/Users/Profile.php

Представления реализованы в виде html шаблонов.

Структура и алгоритм поиска шаблонов на примере шаблона Zero\_Users\_Profile:

1. В области приложений (/application): zero/view/Users/Profile.html
2. В области приложений (/app): view/Zero/Users/Profile.html
3. В области ядра (/phpzero): view/Zero/Users/Profile.html

Оформление располагается в папках (/assets). Структура на усмотрение разработчика.

## Приложение

Алгоритм работы системы в целом описан в компоненте приложение (Zero\_App)..

### Модули (установка не реализовано)

При работе с модулями перечитывается каталог (/application), в котором они находятся.

Справочная информация по модулям читается из их конфигурационных файлов.

Каждый модуль проверяется на статус инсталляции наличием в нем файла:

ModulName/setup/INSTALL (в нем хранится версия установленного модуля).

И выводится в виде списка модулей со следующим функционалом:

Если модуль не установлен его можно установить.

Если он установлен его можно удалить.

*Совет:*

1. Разработку модуля надо начинать с проектирования БД. Не экономьте на этом время.
2. После завершения проектирования в разделе Система - Обслуживания автоматизировано сформируете новый модуль и его модели.
3. На базе сформированных моделей и необходимого функционала (задач) создавайте контроллеры. По необходимости дополняя модуль библиотеками, хелперами.
4. Наиболее удачные модули могут быть включены в официальные модули CMF PhpZero

#### Обновление БД (не реализовано)

В отличие от программной части, которая при обновлении просто копируется. БД так обновлять нельзя по причине наличия данных. Причем обновляемый модуль может отставать на несколько версий.

В каждой из которых могут быть произведены изменения в структуре БД.

То есть необходимо выполнение миграции БД по каждой версии отдельно. Начиная с исходной до обновляемой. Поскольку последнее изменение будет опираться на предыдущую версию БД которая отсутствует в исходном модуле более старой версии.

Поэтому структура и алгоритм обновления БД следующие:

ModulName/setup/schema/

mysql.sql полная схема БД модуля текущей версии.

mysql\_1.1.1.sql миграционный скрипт для указанной версии

mysql\_1.1.2.sql миграционный скрипт для указанной версии

mysql\_1.2.1.sql миграционный скрипт для указанной версии

mysql\_2.4.6.sql миграционный скрипт для указанной версии

*Примечание:*

Известно, что миграция данных очень сложный вопрос, который здесь подробно обсуждать не имеет смысла.

Поэтому выше написанный раздел справедлив для простых случаев и стандартных модулей. В остальных случаях как правило это ручная работа.

mysql\_0.0.0.sql этот скрипт предназначен для удаления модуля из системы (удаление его из БД).

## Архитектура проектирования БД (рекомендации)

### Таблицы

В именах таблиц используются буквенные символы.

Не рекомендуется использование символа нижнего подчеркивания и цифр.

### Свойства

В каждой объектной таблице первым полем всегда идет идентификатор:

(не кросс таблица)

ID

После него (если таковые имеются). Формируются поля связи с другими таблицами.

Правило их именования: (Имя целевой таблицы + “\_” + ID)

GoodCatalog\_ID, Groups\_ID

Для решения конфликтных множественных одноименных связей, имена полей могут быть расширены уникальными однобуквенными суффиксами.

GoodsP\_ID, GoodsC\_ID

*Важно:*

Правило (сигнатура) именования связующих полей строго для соблюдения. Так как абстрактная часть системы повсеместно на это опирается в своей работе. Также на этой архитектуре строятся системы прав.

Далее располагаются все остальные прямые поля (свойства объекта)

Имена этих полей могут состоять только из латинских символов и цифр

Можно использовать верблюжью нотацию.

Name, Description, Price. IsAccess

*Важно:*

Символ нижнего подчеркивания разрешен только в полях связи.

#### Реализация абстрактных горизонтальных прав

Обычные поля связи могут выступать в роли УП если они спроектированы в таблице пользователей.

CarsMarka\_ID (пользователь работает только с моделями определенной марки)

Структура хранения пользователей каталогизирована. Это позволяет реализовывать реальную и необходимую структуру компании в частности и целом. Впрочем, это относится ко всем пользователям проекта. Условие пользователя (если оно установлено) включает самого пользователя и его подчиненных (связанных с ним родительской связью) на один уровень вниз.

Такое решение позволяет организовывать систему прав как между различными подразделениями (группами) по функциональному назначению, так и в внутри одно подразделения дифференцировать согласно ее иерархии.

Тонкости реализации:

CarsMarka\_ID

УП определяется по наличию полей связи в структуре таблицы и статуса УП а также установленным в них значениям.

При обнаружении одноименных полей связи в целевой таблице УП начинают работать.

## Компоненты

Обзор компонентов системы:

1. Приложение ( [Zero\_App](#_toc408) )
2. Система кеширования ( [Zero\_Cache](#_toc498) )
3. Конфигурация ( [Zero\_Config](#_toc544) )
4. Контроллер базовый абстрактный ( [Zero\_Controller](#_toc577) )
5. Нативная работа с БД ( [Zero\_DB](#_toc666) )
6. Объектная работа с БД ( Zero\_AR )
7. Инженеринг ( Zero\_Engine )
8. Фильтр для моделей ( Zero\_Filter )
9. Интернационализация ( )
10. Логирование ( [Zero\_Logs](#_toc876) )
11. Модель базовая абстрактная ( [Zero\_Model](#_toc921) )
12. Сессия ( Zero\_Session )
13. Валидатор свойств моделей ( Zero\_Validator )
14. Представление - шаблонизатор ( [Zero\_View](#_toc1087) )
15. Опции приложения ( Zero\_Option )
16. Ответ. Вывод результатов работы приложения ( Zero\_Response )
17. Запрос. Реализация запросов к сторонним ресурсам ( Zero\_Request )

### Приложение (Zero\_App)

#### Назначение

Используемый класс Zero\_App

Реализует выполнение и управление приложением в целом.

Является его основой и содержит в себе ключевые данные для его работы.

Инициализирует системные константы.

#### Реализация

Браузер:

Точкой входа в приложение (web, api) является файл index.php находящий в корне сайта.

В нем подключается сам компонент.

Далее запускается инициализация приложения и его выполнение.

Инициализация приложения:

Zero\_App::Init()

1. Подключается конфигурация и происходит инициализация программного окружения и глобальных настроек. Прямое назначение свойств компонента (Zero\_Config) значениями из массива конфигурации.
2. Подключение системных и пользовательских функций (…/function.php)
3. Определяется автозагрузчик классов
4. Определяется обработчик исключений
5. Определяется режим работы приложения (Web, Api, Console)
6. Разбирается роутинг, язык, параметры запроса. На основании запрошенного uri
7. Инициализируются компоненты:

Zero\_App::$Config Конфигурация

Site\_Config наследуется от Zero\_Config

Для расширения пользовательской кофнигурации приложения (через композицию)

Zero\_App::$Users Пользователь

Site\_Users наследуется от Zero\_Users

Для расширения пользовательской работы с пользователем (сессия)

Zero\_App::$Section Раздел

Site\_Section наследуется от Zero\_Section

Для расширения пользовательской работы с разделом (страницей) (сессия)

Zero\_App::$Controller Контроллер

Zero\_App::$RouteParams Параметры uri

Zero\_App::$Request Запросы к внешним ресурсам (загрузка реквизитов)

Site\_Request наследуется от Zero\_Request

Для расширения пользовательской работы внешними ресурсами по API (через композицию)

Zero\_App::$Options Опции приложения (загрузка)

Site\_Option наследуется от Zero\_Option

Для реализации пользовательский опций приложения

Zero\_Logs Подсистема мониторинга и профилированного работы приложения

Zero\_DB Работа с БД

Zero\_Cache Подсистема кеширования

Zero\_Session Сессия

Выполнение приложения:

Zero\_App::Execute() (использование БД)

1. Режим обслуживания приложения по IP адресу. Если указаны в конфигурации
2. Ограничение работы приложения по IP адресу. Если указаны в конфигурации
3. Стандартная авторизация через заголовки. Если заданы логи и пароль в конфигурации
4. Инициализация пользователя (Zero\_Users). Если передан токен то по нему.
5. Инициализация запрошенного раздела (страницы сайта) на основе роутинга (Zero\_Section).
   1. Проверка существования (404)
   2. Проверка редиректа (301)
   3. Проверка прав на раздел (403)
   4. Инициализация контроллера
   5. Если контроллер указан проверка прав на действие (403)
6. Инициализация контроллера на основе роутинга. (! Если перед этим раздел не был найден !)
   1. Проверка существования (404)
   2. Проверка прав на действие (403)
7. Если контроллер указан. То инициализация его как исполняемого (Zero\_Controller)
8. Работа контроллера. Выполнение метода действия.
9. Сборка макета страницы и вставка результата работы контроллера. (! если не api !)

Вывод:

Реализован централизованный метод отправки необходимых заголовков браузеру.

И выдача результата в нужном формате (по умолчанию html).

Отдаваемым результатом можно управлять внутри контроллера.

Что и происходит в контроллерах API

Консоль:

Точкой входа для выполнения консольных контроллеров является файл /phpzero/console.php

Инициализация приложения аналогична обычному браузерному приложению.

После нее непосредственно выполняется консольный контроллер.

Либо отрабатывает менеджер запуска консольных задач.

#### Зависимости

Zero\_Config, Zero\_Logs, Zero\_Cache, Zero\_Session, Zero\_Section, Zero\_Users, Zero\_Controller, Zero\_View,

### Кеш (Zero\_Cache)

#### Назначение

Используемый класс Zero\_Cache

Обеспечить увеличение производительности за счет выдачи сформированного ранее и сохраненного контента.

#### Реализация

Инициализация компонента происходит во время инициализации приложения.

Во время инициализации определяется хранилище данных кеша:

1. Хранение данных в файловой структуре
2. Хранение данных в оперативной памяти (Memcache)

Компонент может работать в двух режимах:

1. Через модель, данные которой нужно кешировать.
   1. $Model->CH->Set($index, $data, $time)
   2. $Model->CH->Get($index, $time)
   3. $Model->CH->Reset()
2. Напрямую
   1. Zero\_Cache::Set\_Data($index, $data, $time)
   2. Zero\_Cache::Get\_Data($index, $time)

$index индекс кеша (любая строка)

$data кешируемые данные

$time время жизни кеша в секундах (по умолчанию установлен 0 – вечный кеш)

Реализован механизм связанного или зависимого кеша.

Zero\_Cache::Set\_Link($source\_name, $id);

Сброс сохраняемого кеша при работе со связанным объектом.

Правило формирование индекса:

$Source источник (через модель устанавливается автоматически)

$id идентификатор объекта (через модель устанавливается автоматически)

$index вольная строка (указывается в любом случае)

Примеры:

$Model->Cache->Get(‘Content’, 600);

$Model->Cache->Set(‘Content’, $Value, 600);

Zero\_Cache::Get\_Data(‘Zero\_Section/www/paga/page/url’, 600);

Zero\_Cache::Set\_Data(‘Zero\_Section/www/paga/page/url’, $Value, 600);

Zero\_Cache::Get\_Data(‘Zero\_Section/EnumSet/IsVisible/ru-ru’, 600);

Zero\_Cache::Set\_Data(‘Zero\_Section/EnumSet/IsVisible/ru-ru’, $Value, 600);

*Важно:*

Время жизни кеша нужно указывать и при получении и при его сохранении.

В главном конфигурационном файле можно управлять использованием компонента в целом

#### Зависимости

### Конфигурация (Zero\_Config)

#### Назначение

Используемый класс Zero\_Config

Конфигурирование приложения и инициализация программной среды.

#### Реализация

Инициализация компонента происходит во время инициализации приложения.

Создается объект конфигурации и присваивается в свойству приложения ( $Config ).

В него (его свойства) загружается конфигурация приложения из файла **config.php**

В конце происходит инициализация переменных окружения интерпретатора.

#### Зависимости

Zero\_App, Zero\_Logs

### Контроллер (Zero\_Controller)

#### Назначение

Используемый класс Zero\_Controller

Компонент является базой для создания конкретных контроллеров решающих прикладные задачи (реализующих бизнес логику) проекта. Имеет 3 предопределенных рабочих свойства:

$ModelОбрабатываемая модель (объект).

$View Представление (возвращается контроллером на выходе — в случае Web)

$Params Массив служебных параметров (принимает в него параметры на входе).

Контроллеры делятся на 3 вида:

Api реализуют api запросы к приложению

ничего не возвращают. отдают результат своей работы и завершают работу приложения,

Web реализуют обычное веб представление страницы в браузере

возвращают представление либо результирующую строку,

Console консольные контроллеры реализующие внутреннюю работу приложения.

запускаются по крону и возвращают 0 либо 1

#### Реализация

Централизованно контроллеры создаются через фабрику Factories с сохранением в сессии

Точкой входа (выполнения) контроллера является запрошенный (определенный) метод действия.

По умолчанию и в случае плагинов (они же контроллеры) это метод Action\_Default

Методы Чанки:

Эти методы динамические и защищенные либо приватные.

Реализован (заложен) механизм разделения программного кода контроллера на логически и функционально завершенные блоки кода в отдельные методы (Чанки) и управление ими через методы действия.

Начинаются с префикса Chunk\_

Такие методы должны возвращать булево значение (false or true).

Это позволяет управлять ходом их выполнения.

Примеры таких методов для веб контроллеров:

Chunk\_Init() инициализация входных параметров, какие-то общие проверки

Chunk\_View() формирование общего представления для контроллера в целом

Методы Действия:

Эти методы могут быть статичны.

Это управляющие методы. Реализующие собственно бизнес логику приложения и управления ею. Вызываются централизовано системой либо там где это нужно в других методах действия.

Начинаются с префикса Action\_

Примеры:

Action\_Default() контроллер по умолчанию

Action\_Save() сохранение чего-либо

Action\_Remove() удаление чего-либо

Action\_POST() добавление через api запрос

Внутри выполнения метода можно сохранять сообщения о ходе выполнения его работы.

В дальнейшем их можно вывести пользователю и в профилированные лог файлы. Что и реализовано системно.

$Controller->GetMessage()

$Controller->SetMessage(…)

$Controller->SetMessageError(…)

Возвращаемые значения методами действия:

Api Данные в формате json

Web Объект Zero\_View либо собранный шаблон с данными в виде строки (string)

Console Числа 0 (ок) или 1 (ошибка)

Разработка:

В системе реализованы абстрактные контроллеры для выполнения стандартных операций с данными. (CRUD)

Zero\_Controller -> Zero\_Crud\_Grid -> конкретный контроллер

Zero\_Controller -> Zero\_Crud\_Edit -> конкретный контроллер

Реализована работа по связи один ко многим либо многие ко многим

Режим работы как и другие настройки определяются / переопределяются конкретным контроллером.

Шаблоны контроллеров для разработок лежат здесь:

/phpzero/data

Контроллер, работающий по связи должен располагаться в той же папке, в которой лежат контроллеры работающие с моделью без переходов по связи.

/phpzero/class/Zero/Content

EditLayout.php

GridLayout.php

EditSection.php

GridSection.php

Таким образом, он точно идентифицирует режим работы и необходимые настройки. Поскольку одновременно может существовать контроллер, работающий с теми же объектами без переходов по связи (напрямую).

Edit.php

Grid.php

#### Плагины

Плагины, это простые (методы действия) контроллеры, при своей работе они не сохраняются в сессию централизовано. Их задача реализация вспомогательного функционала и представления для и вокруг целевого контроллера выполняющего основную работу.

При создании плагина вторым необязательным параметром, можно передать дополнительные данные для его работы. Эти параметры сохраняются в служебном свойстве $Params.

Это очень часто нужно и используется при рекурсивном вызове плагинов в представлении. Когда оно собирается. К примеру: пагинатор (постраничный вывод), или комментарии к статье.

В основной своей массе плагины работают от представления через директиву plugin.

Когда оно формирует конечный результат.

#### Зависимости

Zero\_Session, Zero\_I18n, Zero\_App, Zero\_View

### БД (Zero\_DB)

#### Назначение

Используемый класс Zero\_DB

Инкапсулирует в себе всю работу с БД.

Централизованное выполнение запросов.

Мониторинг и снятие статистики их выполнения.

Реализует механизм работы с хранимыми процедурами.

#### Реализация

Инициализация соединения с БД происходит при первом обращении к компоненту (запросе).

Содержит в себе следующие группы методов

* Метод инициализации соединения с БД ( Init ).
* Методы проверок входных данных для запросов (EscI, EscF, EscT, EscS, EscE, EscB, EscD, EscID).
* Методы запросов на получения данных и формирование результата в нужно виде (Select\_).
* Методы запросов на вставку новых данных ( Insert\_ ).
* Методы запросов на обновление данных ( Update\_ ).
* Методы вызова хранимых процедур ( Call\_ ).

*Совет:*

Сложные и уникальные запросы хранятся в статичных методах той модели, к которой преимущественно и относится запрос. Такие методы имеют префикс “DB\_”

#### Зависимости

Zero\_App, Zero\_Logs, Zero\_Helper, Zero\_Cache, Zero\_I18n, Zero\_Model

### Инженеринг (Zero\_Engine)

#### Назначение

Используемый класс Zero\_Engine

Реализует функционал автоматического создания и/или конфигурирования модели (класса) на основании таблицы хранящей ее объекты.

Производится конфигурирование самой модели (определения), ее свойств и связей.

Развернуто конфигурирует свойства. Создает конфигурацию для фильтров, табличного представления, представления в формах и работы с БД.

#### Реализация

На входе фабрика принимает группирующий префикс таблиц()ы.

Производится анализ соответствующей таблиц(ы) и на основе его результата конфигурируется модель (модели).

Шаблон для создания модели:

/zero/data/Template\_Model.php

Также формируется файл интернационализации из шаблона:

/zero/data/Template\_I18n.php

Также производит первичное создание модуля.

Создается файловая структура, главный раздел.

Дополнительную информацию и возможностях можно почерпнуть из самого компонента.

Реализованные формы:

Формы и типы свойств (определяемые при анализе):

Number - (text) Число

Text - (text) Строка текста

Select - (select) Перечисление

Radio - (radio) Перечисление с умолчанием (определяет по не нулевому значению поля Enum)

Checkbox - (checkbox) Множество

Textarea - (textarea) Текст

Date - (text) Дата

Time - (text) Время

DateTime - (text) Дата и время

Link - (select) Свойство связи с другим объектом

Формы и типы свойств (не определяемые при анализе):

Hidden - (hidden) Скрытое поле

ReadOnly - (-) Только для чтения

Password - (password) Строка пароль

File - (file) Ссылка на файл (пытается определить по имени поля)

FileB - (file) Бинарные данные файла (пытается определить по имени поля и типу)

Img - (file) Ссылка на картинку (пытается определить по имени поля)

ImgB - (file) Бинарные данные картинки (пытается определить по имени поля и типу)

Content - (textarea) Большей текст (визевик) (пытается определить по имени поля)

LinkMore - (select) Свойство связи с другим объектом (многочисленным - ajax)

#### Зависимости

Zero\_DB, Zero\_Logs, Zero\_Model, Zero\_App, Zero\_Helper, Zero\_I18n

### Фильтр (Zero\_Filter)

#### Назначение

Используемый класс Zero\_Filter

Формирование фильтров, условий поиска, сортировок и деления на страницы для получения определенного среза данных.

#### Реализация

Инициализируется по необходимости в конкретном контроллере.

Работает в тесной связке с моделью, для которой и формирует фильтр.

Формируется через статичный фабричный метод и сохраняется в сессии, принимая в качестве аргумента целевую модель. Формирование фильтра:

Zero\_Filter::Factory($Model);

Передается компоненту работы с БД, который централизованно формирует условия для последующего запроса к источнику.

Фильтр содержит в себе:

Фабричный метод формирования фильтра

Методы добавления фильтров (их настроек), установки их значений и их получение.

Методы добавления полей для поиска, установки их значений и их получение.

Методы добавления полей для сортировок, установки их значений и их получение.

Метод сброса фильтра в целом или для определенного свойства.

Конфигурация фильтра заложена в статичном методе модели, для которой он формируется.

Config\_Filter()

Вызывается конфигурация через динамический метод базовой модели.

Get\_Config\_Filter()

По умолчанию фильтры строятся для полей:

Перечисления (Select, Radio)

Дата и время (DateTime)

Поля связи (Link, LinkMore)

Множества (CheckBox)

По умолчанию поиск и сортировка формируется для всех остальных числовых и строковых полей (поля в фильтрах исключаются).

Эти настройки можно переопределять по необходимости.

*Важно:*

Персональные фильтры свойств реализуются в модели, к которой они относятся.

Имена таких методов должны начинаться с префикса “ FL\_”

#### Зависимости

Zero\_App, Zero\_Model, Zero\_Session

### Интернационализация (Zero\_I18n)

#### Назначение

Перевод контента на нужный язык. Интернационализация проекта.

Используемый класс Zero\_I18n

Специальной инициализации не требуется

Системно переводит:

Название контроллеров и их действий (методов)

Сообщения действий контроллеров.

Название моделей и их свойств.

Значение для полей Enum и Set

Сообщения ошибок валидации свойств.

#### Реализация

Переводы реализованы с помощью одномерных ассоциативных массивов находящихся в языковых файлах.

Эти массивы являются возвращаемым значением этих файлов.

Таким образом, поиск нужно перевода производится по языковому файлу и ключевой фразе (индексу).

Поиск файла перевода происходит аналогично поиску класса с моделью с той лишь разницей, что поиск ведется в папке “i18n” модуля, а не “class”

Info\_Article

1. /application/Info/i18n/Article.php
2. /zero/i18n/Article.php
3. /application/Info/i18n/App.php
4. /zero/i18n/App.php

Если указан файл Info\_App то поиск идет напрямую только этого файла:

1. /application/Info/i18n/App.php
2. /zero/i18n/App.php

Этот языковой файл (его имя) зарезервирован для перевода общих и ключевых элементов.

(Название проекта, Имя почтового адреса по умолчанию для всего приложения, …)

Это реализовано для удобства. Вы вправе сами решать, где и какой перевод хранить.

Если проект маленький можно весь перевод хранить в одном файле (Www\_App)

Далее в полученном массиве из файла производится поиск элемента с нужной ключевой фразой (индексом)

Если по ключевой фразе перевод не был найден, к ней добавляется приставка “translation ” и поиск повторяется. Пример перевода:

$translation = Zero\_I18n::T(‘Info\_Article’, ‘translation Head’, ‘Head’);

Файлы перевода формируются на основе моделей или функционала, для которых нужен перевод.

Info\_Article

Ключевые фразы формируются с использование в начале ключевого слова “translation ”

“translation Head”

Зарезервированные системные ключевые фразы:

controller ControllerName => Имя контроллера

controller ControllerName action ActionName => Название действия

controller ControllerName message Key => Сообщение на действие

model => Имя модели

model prop PropName => Название свойства модели

model prop PropName validate Key => Сообщение валидации свойства

model prop PropName option Key => Варианты значений для свойств ENUM, SET

Для удобства переводов внутри html шаблонов представления реализована директива translation.

#### Реализация переводов в БД

В системе используется асинхронный подход к реализации переводов.

Структура в таблице:

News(ID, Language\_ID, Name, …)

В том числе и для требующих синхронного подхода.

В БД для таких объектов создаются отдельные языковые таблицы (синхронный перевод):

News(ID, Name), NewsLanguage(ID, News\_ID, Language\_ID, Name)

Но на уровне приложения подход также остается асинхронным. А именно:

Создаются два отдельных объекта $News и $NewsLanguage.

С которыми в дальнейшем можно работать как с единым целым, так и по отдельности.

$News использовать для функционала (какие либо вычисления) требующего точной идентификации объекта как целого.

$NewsLanguage использовать в информационных целях и дальнейших переводах как отдельный объект конкретного языка.

Такой подход упрощает реализацию и понимание приложения, а также добавляет ей гибкости.

#### Зависимости

Zero\_App, Zero\_Logs

### Логирование (Zero\_Logs)

#### Назначение и Описание

Используемый класс Zero\_Logs

Реализует функционал профилированный мониторинг и снятие статистических данных о работе приложения в частности и в целом. Профилирование работы приложения.

#### Реализация

Инициализация компонента происходит во время инициализации приложения.

Фиксирует затраченное время на выполнение приложения в целом и по его составным частям.

Затраченной оперативной памяти.

Фиксирует ошибки различного уровня в работе приложения и запросов к БД.

Действия пользователя и их результат.

Фиксирует работу фоновых задач приложения.

Выводит отладочную информацию в браузер.

Формирование статистики происходит при завершении работы приложения (происходит перехват).

Является обработчиком исключений и ошибок программирования

Обрабатывает в исключениях ошибки 403, 404, 409, 500.

Использование таймеров

Zero\_Logs::Start('комментарий');

замеряемый код

Zero\_Logs::Stop('комментарий');

Использование сообщений (ошибки, предупреждения, отладочная информация)

Zero\_Logs::Set\_Message\_Error('Логируемое сообщение');

Zero\_Logs::Set\_Message\_Warning('Логируемое сообщение');

Zero\_Logs::Set\_Message\_Notice('Логируемое сообщение);

Zero\_Logs::Set\_Message\_Info('Логируемое сообщение');

Логи сохраняются централизовано при вызове завершающего и выдающего результат работы метода приложения (Zero\_Response)

Разработка и отладка

Частное или локальное логирование

Zero\_Logs::Custom($logFileName, $logData);

Zero\_Logs::Custom\_DateTime($logFileName, $logData);

Использование исключений рекомендуется использовать только для отладки.

throw new Exception('metod not found: ' . get\_class($this) . ' -> ' . $method, 409);

throw new Exception('Not Found', 404);

#### Зависимости

Zero\_App, Zero\_View, Zero\_Session

### Модель (Zero\_Model)

#### Назначение

Используемый класс Zero\_Model.

Реализует основную работу с моделями. Являясь базовым абстрактным классом для каждой из них.

Взаимодействие с данными хранящимися в БД. Предоставление доступа к ней для контроллеров.

#### Реализация

Создание:

Модель создается и инициализируется через фабрику и фабричный метод:

1. Создание через фабричный метод (Make), с сохранением в сессию (Factory, ...).
2. Создание через фабрику (Makes) Фабрика производит поиск и создание модели соответствующей переданному классу.

При указании флага загружает модель из БД под указанным ID.

Реализован механизм динамического сохранения моделей в сессии (Factory\_Set, Factory\_Get)

Работа со свойствами:

Реализована абстрактная работу со свойствами моделей через магические методы и их конфигурации в конкретной модели:

Config\_Prop Полная конфигурация всех свойств модели и их атрибутов.

Config\_Grid Конфигурации свойств для вывода данных в виде различных списков

Config\_Filter Конфигурации свойств используемых для фильтрации получаемых данных

Config\_Form Конфигурации свойств используемых в различных формах редактирования

Эти методы принимают в себя рабочую модель и сценарий (имя контроллера обработчика).

Благодаря этому можно управлять получаемым набором свойств и их порядком.

Также учитывать и реализовывать права на них.

Изменение свойств производится как обычно:

$News->Description = $value;

Реализован механизм хранения состояния свойств модели во время работы.

Что позволяет оптимизировать работу с БД. Производить обращение к ней, только если свойства действительно были изменены.

Для комплексной работы со свойствами внутри модели реализованы следующие методы:

$this→Set\_Props($properties) Изменение свойств.

Если $properties пустой то статус всех свойств модели выставляется как актуальный (не измененный)

$this→Get\_Props($flag) Получения свойств с определенным статусом

$flag = -1 получить только измененные свойства и их значения

$flag = 0 получить все свойства (по умолчанию) и их значения

$flag = 1 получить только не измененные свойства и их значения

Инструменты (служебные методы и компоненты) модели:

Используется алгоритм ленивой загрузки компонентов.

Компоненты инициализируются при обращении к специализированным свойствам, сохраняющих их.

В момент обращение к компоненту через свойство ему делегируется текущая модель.

И далее с ним производится необходимая работа. Имена таких свойств состоят из двух заглавных букв

Имена сопутствующих служебных методов начинаются с соответствующих заглавных букв.

Валидация ( Zero\_Validator )

Валидаторы свойств имеют вид

$News→VL\_PropertyName($value, $scenario)

Валидация свой вызывается методом

$News→VL→Validate($data, $scenario)

Фильтры ( Zero\_Filter )

$News→FL\_PropertyName() формирование и получение списка допустимых значений для целевого свойства

Кеширование ( Zero\_Cache )

$News→CH→Get($index, $time) Получение кешированных данных

$News→CH→Set($index, $time) Создание, сохранение данных в кеш

$News→CH→Reset() Сброс кеша объекта

Объектная работа с БД ( Zero\_AR )

$News→AR\_From($params) Методы реализующие объектное формирование запроса к БД

$News→AR\_NameMethod($params)

Нативная работа с БД ( Zero\_DB )

$News→DB\_NameMethod($params) Методы реализующие прямые запросы к БД (могут быть статичные)

*Примечание:*

Используется паттерн композиция с делегированием при работе с компонентами.

Более подробную информацию о работе с моделью и ее компоннетами смотрите в документации кода.

#### Зависимости

Zero\_App, Zero\_Session, Zero\_I18n, Zero\_DB, Zero\_Validator, Zero\_Cache, Zero\_Logs

### Сессия (Zero\_Session)

#### Назначение

Используемый класс Zero\_Session.

Наследуется от библиотечного класса ArrayObject

Централизованное сохранение необходимых данных (прежде всего объектов) в сессию.

Для доступа к ним между процессами (запросами).

#### Алгоритм работы

Инициализация компонента происходит во время инициализации приложения

Создает единственный экземпляр (объект) самого себя внутри своего статичного свойства.

И в дальнейшем вся работа идет через него.

Таким образом, реализует сохранение любых данных в виде индексированного массива внутри себя.

Все рабочие методы компонента статичные

#### Зависимости

Zero\_Logs

### Валидатор (Zero\_Validator)

#### Назначение

Используемый класс Zero\_Validator

Проверка данных пришедших извне на правильность заполнения. Формализация их до нужного вида.

#### Реализация

Инициализация компонента происходит при обращении (использовании) через свойство конкретной модели.

Проверка может быть как по форме представления свойства, так и по его названию. Возможно использование сценариев проверки. Реализует механизм сообщений с детализацией по каждому проверяемому свойству.  
Методы проверки свойств (валидаторы) могут быть расширены моделью, свойства которой проверяются.

Работает только через модель (его свойство).

Запуск и выполнение валидации (проверки):

$Model->Validator->Validate($\_REQUEST['Prop']);

Порядок поиска:

1. Поиск по имени свойства в самой модели. В качестве аргументов таким методам передается проверяемое значение и сценарий.
2. Поиск по форме представления в валидаторе. В качестве аргументов таким методам передается проверяемое значение и целевое свойство.

Если ничего не найдено значение присваивается свойству, безусловно.

Непрошедшие проверку свойства и их причины заносятся в массив:

$Validator->Errors;

И доступны через метод

$Validator->Get\_Errors();

Конфигурация для первичной валидации и поиска валидаторов по форме представления заложена в статичном методе модели.

Config\_Form()

Вызывается конфигурация через динамический метод базовой модели.

Get\_Config\_Form()

*Важно:*

Персональные валидаторы свойств реализуются в модели свойства которой они валидируют.

Имена таких методов должны начинаться с префикса “VL\_”

*Примечание:*

Также реализован поиск начального и завершающего метода валидации, с передачей им всего массива пришедших данных.

$Validator->Validate\_Before ($data, $scenario);

$Validator->Validate\_After ($data, $scenario);

#### Зависимости

Zero\_Model, Zero\_Helper, Zero\_Logs, Zero\_I18n

### Представление (Zero\_View)

#### Назначение

Используемый класс Zero\_View

Представление решает задачу разделения представления информации от ее формирования.

Позволяет разделить работу программиста и верстальщика.

Тем самым ускоряет процесс верстки шаблонов и интерфейсов представления информации.

А также позволяет программисту сконцентрироваться на бизнес логике приложения, не отвлекаясь на верстку. Благодаря параллельной работе ускоряется процесс разработки приложения.

Представление реализовано шаблонизатором.

Специальной инициализации компонент не требует.

#### Реализация

Организация расположения шаблонов уже описана в разделе [4.3](#_toc211)

##### Комментарии

Различные технические комментарии. Удаляться при компиляции шаблона.

{# comment\_name #}

##### Переменные

Вывод значение переменной ( к примеру название товара )

{$name}

Присвоение нового значения переменной без вывода ( к примеру счетчик )

{set $price += 5}

##### Функции

Перевод каретки (\n) меняет на <br>\n

{nl2br($name)}

Перевод html символов в символы их дословного отображения

{htmlspecialchars($name)}

Вывод структуры и данных указанной переменной

{pre($data)}

Вывод списка всех переданных переменных (данных) в шаблон

{pre($\_)}

Вывод всех переданных переменных (данных) в шаблон (их значений)

{pre($\_\_)}

*Примечание:*

Собственно можно использовать любую встроенную или пользовательскую php функцию

##### Логика

Вывод, обработка содержимого блока при истинности условия в выражении

{if ( $count > 20 )} ... {/if}

Вывод, обработка содержимого одного из 2 блоков в зависимости от истинности условия в выражении

{if ( $count > 20 )} ... {else} ... {/if}

Вывод, обработка содержимого одного из блоков в зависимости от истинности стоящего перед ним условия в выражении

{if ( $count > 20 )} ... {else if ( $count > 10 )} ... {else} ... {/if}

Логический переключатель. Целесообразно использовать, когда вариантов ветвления более 5

{switch ( $expression )}

{case $value:} <span>popcorn</span> {break}{/case}

{case $value:} <span>octopus</span> {break}{/case}

{default :} <span>octopus</span> {/default}

{/switch}

*Важно:*

Первый case пишется сразу же после указания switch: {switch ( $expression )}{case $value:} …

##### Циклы

Формирование списка, таблицы из массива элементов ( к примеру товары, группы, статус чего либо ) если он не пустой ( хотя бы один элемент )

{foreach ($wares as $id=>$ware)}

... <a href="index.php?id={$id}">{$ware['name']}</a> ... {$ware['description']} ...

Или

... <a href="index.php?id={$id}">{$ware}</a> ...

{/foreach}  
...

{foreach ($status\_list as $value)}

... {$value}<br> ... Или ... {$value[‘status’]}<br> ...

{/foreach }

...

{for ($i=0; $i < count($wares); $i++)}

номер {$i} <a href="index.php?id={$wares[$i]['id']}">{$wares[$i]['name']}</a>

{/ for }

...

Выполнение блока при соблюдении условия в выражении

{while ( $count > $count\_iteration )}

... {$name} ...

{set $count\_iteration+=1}

{/while }

*Примечание:*

{/if}, {/foreach }, {/for}, {/while}, {/switch}, {/case}, {/default}

Вместо них можно использовать один для всех элементов {/}

Для управления ходом выполнения циклов и переключателя есть возможность использовать следующие конструкции: {break}, {continue}

##### Директивы

**Перевод:** {**translation** “Zero\_Users” ”key”} или {**lang** “Zero\_Users” ”key”}

Перевод ключевого слова на текущий язык. Производится во время компиляции шаблона.

На выходе формируется языковой шаблон с нужным переводом.

Алгоритм поиска и перевода описан в компоненте Zero\_I18n

**Плагин:** {**plugin** ”plugin\_name” [property\_name1=value1 property\_name2=value2]}

Подключение и выполнение плагина в шаблоне.

*plugin\_name* – имя плагина в формате имени контроллера

*property\_name1, property\_name2* – имена свойств плагина

*value1, value2* – соответствующие значения для свойств. Значение может быть любое.

**Включение:** {**include** “Zero/Include/Headermeta”} или {**inc** “Zero/Include/Headermeta”}

Включение шаблонов *path\_to\_filename* в текущий шаблон как есть без компиляции.

**Литерал:** {literal} ………. {/literal }

Исключение заключенного содержимого между этими тегами для обработки. Оставляется как есть..

Для решения конфликтов с клиентскими или иными языками программирования.

##### Пути

{HTTP} Ссылка на главную страницу ( http://www.domain.com)

{HTTPH} Полная ссылка на страницу с которой пришли ( http://www.domain.com/page/page/.... )

{HTTPA} http ссылка на статичные данные (images, css, js) ( http://www.domain.com/assets)

{HTTPD} http ссылка на бинарные данные объектов и пользователя (http://www.domain.com/upload/data)

{URL} относительная ссылка на запрошенный раздел в системе ( /page/page/page ) с учетом языка

##### Примеры

Пример абстрактного использования:

<input type="text" id="{$prop}" name="Prop[{$prop}]" value="{htmlspecialchars($\_REQUEST['Prop'][$prop])}">

<input type="text" id="{$prop}" name="Prop[{$prop}]" value="{htmlspecialchars($Client->$prop)}">

Пример элемента формы:

<input type="text" id="Name" name="Prop[Name]" value="{htmlspecialchars($\_REQUEST ['Prop']['Name'])}">

<input type="text" id="Name" name="Prop[Name]" value="{htmlspecialchars($Client->Name)}">

Правило формирование элемента формы:

name="Prop[Name]"

Служит для централизованной обработки формы системой.

Prop – группирующий индекс для свойств конкретного объекта.

Name – Имя (свойства, поля, переменной).

value="{htmlspecialchars($\_REQUEST ['Prop']['Name'])}"

Содержит ранее введенные данные в текущее поле. (Для добавления новых элементов – регистрация клиента, новое сообщение…)

value="{htmlspecialchars($Client->Name)}"

Содержит данные редактируемого объекта. (Для изменения существующих элементов – профиль клиента, сделанные заказы…)

Там где это необходимо (в контроллерах) создается объект представления.

При создании в конструктор передается конкретный шаблон.

$View = new Zero\_View(‘template\_name’);

Шаблоны можно добавить позднее в стек.

$View->Add(‘template\_name’);

Или удалить из стека.

$View->Delete(‘template\_name’);

Далее в него передаются данные для вывода.

$View->Assign('variable', 'value');

После чего происходит вызов на выполнение шаблона и получение готового результата.

$View→Fetch(); для html шаблонов

*Примечание:*

$View->Fetch(); вызывается системно самим приложением.

Все бинарные данные хранятся в каталоге /upload.

#### Зависимости

Zero\_App, Zero\_Logs, Zero\_Helper

### Запросы к внешним ресурсам (Zero\_Request)

#### Назначение

Используемый класс Zero\_Request производный (наследуемый) Site\_Request и композиции

Реализует обертку для реализации запросов по API к внешним сервисам и ресурсам.

Site\_Request - Является контейнером для реализации пользовательских оберток к сторонним ресурсам и сервисам. Через паттерн композиция. А также возможного переопределения базового функционала.

#### Алгоритм работы

Инициализация происходит во время инициализации приложения. (Zero\_App::$Request)

Имя вызываемого метода или свойства (композиция) реализующего запросы к конкретному ресурсу, является ключом секции настроек запроса в конфигурационном файле (AccessOutside).

Оно также служит суффиксом файл лога его работы.

Настройки соответственно забираются из конфигурации в момент первого запроса (ленивая загрузка).

Важно:

Это ключевое абстрактное понятие Для реализации пользовательских оберток:

имя ключа в конфигурации = имя метода или свойства = [суффикс пользовательского класса обертки]

( Billing = ->Billing(…) || ->Billing->… = Site\_Request\_Billing )

Запросы к родственному ресурсу реализованы в самом компоненте абстрактно через метод перегрузки.

(\_\_call)

Внутри метод производит всю черновую работу:

Подготовка данных для запроса (json), сам запрос с отправкой нужных заголовков.

Получение и обработка ответа (ошибки).

Инициализация полученных данных в рабочий массив данных, готовый к дальнейшей работе

Возврат в точку вызова этих самых данных.

Формат передачи данных между ресурсами происходит в формате json

Пример:

$response = Zero\_App::$Request->Calculator('POST', "/api/v2/calc/custom", ['groups' => $gr, 'location' => $lc]);

Для реализации запросов к сторонним ресурсам со своей спецификацией создается отдельный класс, реализующий всю спецификацию нужных запросов и их вызовов (пример Zero\_Request\_Billing)

Инициализация и сохранение в компоненте (Zero\_Request) происходит один раз в момент первого вызова.

Доступ к этому свойству осуществляется через метод перегрузки.

(\_\_get)

Пример:

$response = Zero\_App::$Request->BillingNL->GetClients();

Возвращаемое значение это объект со свойствами:

$Head Заголовки ответа

$Body Тело ответа целиком

$Code Код ответа (json - phpzero)

$Message Информационное сообщение (json - phpzero)

$Content Тело ответа (json - phpzero)

$Error Признак ошибки (json - phpzero)

#### Зависимости

Нет

# Справочник (Reference)

## Приложение

Работа программного комплекса в целом над конкретным проектом. А также компонент Zero\_App в частности, который осуществляет управление его работой.

## Система, ядро

Совокупность программного обеспечения необходимая для работоспособности программного комплекса в целом (/phpzero).

## Модуль

Совокупность программного обеспечения (контроллеры, модели, представления …) решающие какую либо конкретную прикладную задачу.

## Плагин

Это частный случай контроллера, решающий вспомогательные задачи. Как правило на веб страницах.

## Хелпер

Набор однотипных и общих для большинства задач вспомогательных функций, объединенных в класс.

Все методы такого класса как правило статичные. Вызываются по мере необходимости решения конкретных задач.

## Компонент, подсистема

Это логически законченный функционал решающий системные задачи.

Компоненты реализованы классами.

## Макет (layout)

Это главный шаблон представления, в который вставляются все сформированные ранее данные.

## Представления

Это частные шаблоны конкретного контроллера либо шаблоны макетов всей страницы.

## Чанки (Chunk)

Это логически и функционально завершенные блоки кода контроллеров заключенные в отдельные методы.

(Чанк инициализации, чанк формирования вывода, чанк выполнения действий к.)

## Двоечник(и)

Это таблица связей многие ко многим. Или кросс таблица

## Свойство

Применяется в широком смысле слова: свойство объекта, полей в таблице БД, поле в форме представления.

## Компиляция шаблона

Анализ и замена в html шаблонах php псевдо кода на синтаксически правильный php код.

## Источник

Под источником понимается абстрактное хранилище конкретных данных.

Это может быть таблица в реляционной БД, или коллекция в Mongo DB, просто файл.

На данный момент это только MySQL

## Группа

Под группой в данной системе понимается не только объединение определенных пользователей.

Но также их права, как вертикальные, так и горизонтальные.

## УП

Условия пользователя накладываемые в процессе работы контроллеров.

Реализуют горизонтальные права доступа к рабочей информации.

## УС, УРС, УКС

Условие связи. Определяет понятие связи одного объекта с другим и его тип.

УС - условие связи один ко многим и общий термин обозначающий связь (зависит от контекста)

УРС – условие рекурсивной связи или связи на себя. Каталогизированные объекты.

УКС – условие связи многие ко многим

# API

## Адрес запроса и нахождение контроллера

Методы: [GET|PUT|POST|DELETE|OPTIONS]

Основной вариант

<http://domain.com/api/v1/page/page/page?param=value&param1=value1>

api признак api запроса

v1 версия

page/page/page urn адрес контроллера

Альтернативный вариант (нативный) (доступный всегда если контроллер реально существует)

<http://domain.com/api/v1/ControllerName/ControllerName/ControllerName/page?param=value&param1=value1>

ControllerName/ControllerName/ControllerName Имя контроллер

Во время поиcка контроллера разделитель “/” заменяется на “\_ “

Пример:

Чтобы сделать запрос к контроллеру Domain\_Api\_Basket

Необходимо выполнить следующий запрос:

<http://domain.ru/api/v1/Domain/Api/Basket?params>....

Данные запроса передаются в формате json

Ответ приходит также в формате json и имеет следующий вид

{

"Code": 0, код сообщения (0 любой правильный ответ, -1 любой не правильный ответ)

"Message": "", текст сообщения

"Error": true|false, признак ошибки

"Content": … Тело ответа

}

## OPTIONS

Метод описывает реализованные методы запроса.

Запрос без параметров возвращает информацию о реализованных методах.

OPTIONS: /api/v1/page/page

При запросе конкретного метода возвращается подробную информация о методе запроса.

OPTIONS: /api/v1/page/page?POST=1

Формат ответа:

{

"Name": “Описание запроса”,

"Uri": "Параметры адресной строки",

… - Данные или тело запроса (для запросов POST & PUT)

}