ClickBlocks\MVC\Controller

General information

Inheritance	no
Subclasses	no
Interfaces	no
Source	Framework/mvc/controller.php

Controller plays the role of the front controller in MVC model. It is a centralized entry point for handling all HTTP requests to the site pages. Using this class you can assign to each request any function or object that supposed to be able to manage this request. As a rule, as such objects the class **ClickBlocks\MVC\Page** is used. Mapping between HTTP requests and page classes carried out by the same rules as the creation of routing map for **Clickblocks\Net\Router**. After the page object is obtained, **Controller** starts performing of the workflow for that object, i.e. invokes some its methods in strictly certain order. The example below shows how to launch workflow of the requested page:

```
function foo($param) {...}

$test = function() {...}

// Creating routing map.
$map = [''           => ['GET|POST'        => ['callback' => 'ClickBlocks\MVC\Main[]']],
        'users'      => ['GET|POST'        => ['callback' => 'ClickBlocks\MVC\Main:users']],
        'users/#ID#' => ['GET|POST|DELETE' => ['callback' => 'ClickBlocks\MVC\User[1]']],
        'foo'        => ['GET'             => ['callback' => 'foo']],
        'test'       => ['PUT'             => ['callback' => $test]]];

// Performs workflow.
(new Controller($map))->execute();
```

Also with this class you you will be able to temporarily restrict the access to your site if you wish and set up error handler for the following two situations: when the requested page is not found and when access to the requested page is denied.

```
(new Controller($map))
->setErrorHandler(404, function(){echo 'Page not found';})
->setErrorHandler(403, function(){echo 'Access denied';})
->execute();
```

Public static properties

cache

```
public static ClickBlocks\Cache\Cache $cache
```

An instance of cache object which can be used for locking/unlocking of the application.

Public non-static methods

__construct()

```
public void __construct(array $map = [], Cache\Cache $cache = null)
```

$map	array	routing map, used for setting up mapping between HTTP requests and page classes.

$cache	ClickBlocks\Cache\Cache	an instance of cache object that cen be used for locking/unlocking of the application.

The class constructor checks whether the application is locked or not. If it is, the mark of site locking will be stored in the cache and the appropriate message will be shown. There are three ways to set up cache object for storing locking mark (listed in order of decreasing priority):

1. via the second parameter of the constructor;
2. via the static property **$cache**;
3. by using the default cache object.

Managing of locking/unlocking is carried out by changing the appropriate configuration parameters. (See more details in the tutorial).

setErrorHandler()

```
public self setErrorHandler(integer $status, mixed $callback)
```

$status	integer	HTTP status code. Only status code 404 and 403 can be processed.
$callback	mixed	any valid delegate string or delegate object.

Sets up an HTTP error handler for processing one of the two permitted HTTP errors.

getErrorHandler

```
public ClickBlocks\Core\Delegate|boolean getErrorHandler(integer $status)
```

$status	integer	HTTP status code. Only status code 404 and 403 can be processed.

Returns the previously defined HTTP error handler or FALSE if a handler for such status code is not yet defined.

execute()

```
public mixed execute(ClickBlocks\MVC\Page $page = null, string|array $methods = null, string|ClickBlocks\Net\URL $url = null)
```

$page	ClickBlocks\MVC\Page	an instance of page class. If this parameter is defined, the workflow process will be applied to it and otherwise it will be applied to the page class object that determined via routing map.
$methods	string, array	HTTP request methods. You can use this parameter in order to manually define the necessary HTTP method or methods for processing.
$url	string, ClickBlocks\Net\URL	some url string or object. Using this argument you can redefine URL of the current HTTP request.

You can use this method in three basic cases:

1. When no parameters are defined. This is the most widespread case. The page object is created according to routing map and parameters of the current HTTP request. After creation page object, the workflow will be applied to it.
2. When the first parameter is only defined. In this case, workflow is applied to the page object specified as the first parameter of the method. In can be useful for performing workflow of some existing page object. The following example shows how you can render HTML of the some existing page (providing that the current HTTP request is GET):

```
// Creating our page object.
$page = new MyPage();
// Creating controller object.
$controller = new Controller($map);
// Starts buffering.
ob_start();
// Performs workflow for our page.
$controller->execute($page);
// Ends buffering and gets HTML of the page.
```

```
$html = ob_get_clean();
```

3. When the second and third parameters are only defined. In this case, the page object is created from routing map according to given HTTP methods and/or URL. For example:

```
// Creating controller object.
$controller = new Controller($map);
// Creating page object from routing map according to given HTTP method and URL
// with subsequent performing of workflow.
$controller->execute(null, 'GET', '/my/page/url');
```

The method returns nothing if the request mapped to a page object. Otherwise, it returns result of performing of the mapped function.

Protected non-static properties

map

```
protected array $map = []
```

Routing map for mapping between HTTP requests and page class objects.

handlers

```
protected array $handlers = []
```

Array of the basic HTTP error handlers.

> Only HTTP errors with status codes 404 and 403 can be processed.