Controllers

Controllers are part of the MVC architecture. They are objects of classes extending from yii\base\Controller and are responsible for processing requests and generating responses. In particular, after taking over the control from applications, controllers will analyze incoming request data, pass them to models, inject model results into views, and finally generate outgoing responses.

Actions

Controllers are composed of actions which are the most basic units that end users can address and request for execution. A controller can have one or multiple actions.

Controller Lifecycle

When processing a request, an application will create a controller based on the requested route. The controller will then undergo the following lifecycle to fulfill the request:

1. The yii\base\Controller::init() method is called after the controller is created and configured.
2. The controller creates an action object based on the requested action ID:
    - If the action ID is not specified, the default action ID will be used.
    - If the action ID is found in the action map, a standalone action will be created;
    - If the action ID is found to match an action method, an inline action will be created;
    - Otherwise an yii\base\InvalidRouteException exception will be thrown.
3. The controller sequentially calls the `beforeAction()` method of the application, the module (if the controller belongs to a module), and the controller.
    - If one of the calls returns `false`, the rest of the uncalled `beforeAction()` methods will be skipped and the action execution will be cancelled.
    - By default, each `beforeAction()` method call will trigger a `beforeAction` event to which you can attach a handler.
4. The controller runs the action.
    - The action parameters will be analyzed and populated from the request data.
5. The controller sequentially calls the `afterAction()` method of the controller, the module (if the controller belongs to a module), and the application.
    - By default, each `afterAction()` method call will trigger an `afterAction` event to which you can attach a handler.
6. The application will take the action result and assign it to the response.

Best Practices

In a well-designed application, controllers are often very thin, with each action containing only a few lines of code. If your controller is rather complicated, it usually indicates that you should refactor it and move some code to other classes.

Here are some specific best practices. Controllers

- may access the request data;
- may call methods of models and other service components with request data;
- may use views to compose responses;
- should NOT process the request data - this should be done in the model layer;
- should avoid embedding HTML or other presentational code - this is better done in views.