## Жизненный цикл приложения ASP.NET MVC

Отправляя из браузера запрос к приложению, нередко мы практически моментально получаем нужный контент. Однако, в реальности обработка запроса проходит кучу различных этапов. И в этой статье мы рассмотрим основные этапы жизненного цикла приложения на ASP.NET MVC 5.

1. После получения IIS запроса на обработку, при первом обращении к ресурсу, работающему под управлением CLR, создается объект класса ApplicationManager, который представляет домен приложения, внутри которого обрабатывается запрос. Домены приложений изолируют выполняющиеся приложения друг от друга. Уже внутри домена приложения создается объект класса HostingEnvironment, который предоставляет доступ к информации о приложении, в частности, он сообщает имя и каталог приложения.
2. После создания домена приложения также создаются и инициализируются такие объекты, как HttpContext, HttpRequest и HttpResponse, которые инкапсулируют всю информацию, связанную с текущим запросом к приложению.
3. На следующей стадии уже непосредственно запускается приложение, которое представляет экземпляр класса HttpApplication. Если в приложении определен файл Global.asax, то среда ASP.NET в качестве приложения создает объект класса Global.asax, который в свою очередь наследуется от класса HttpApplication. На этой же стадии происходит начальная инициализация приложения в методе Application\_Start, который находится в файле Global.asax.cs.
4. После этого запрос начинает обрабатываться в конвейере класса HttpApplication - начинается собственно обработка запроса приложением. И первым шагом здесь является установка маршрута. Запрос перехватывается специальным HTTP-модулем под названием UrlRoutingModule. Этот модуль выбирает маршрут, который соответствует входящему запросу. Весь набор маршрутов определяется в файле RouteConfig.cs

public class RouteConfig

{

    public static void RegisterRoutes(RouteCollection routes)

    {

        routes.IgnoreRoute("{resource}.axd/{\*pathInfo}");

        routes.MapRoute( name: "Default",

            url: "{controller}/{action}/{id}",

            defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }

        );

    }

}

Сама регистрация маршрутов происходит в методе Application\_Start строкой RouteConfig.RegisterRoutes (RouteTable.Routes);. Для каждого приложения имеется только один объект коллекции маршрутов RouteTable.

1. Если модулю UrlRoutingModule удастся сопоставить запрос с одним из маршрутов в коллекции RouteTable, то затем UrlRoutingModule выбирает обработчик маршрутов сопоставленного маршрута - объект IRouteHandler. По умолчанию объект IRouteHandler представляет экземпляр класса MvcRouteHandler.
2. Затем у объекта IRouteHandler вызывается метод GetHttpHandler, который возвращает объект интерфейса IHttpHandler, используемый для обработки запроса. По умолчанию в качестве IHttpHandler-a используется объект класса MvcHandler.
3. У обработчика IHttpHandler вызывается метод ProcessRequest для обработки запросов:

protected internal virtual void ProcessRequest(HttpContextBase httpContext)

{

    SecurityUtil.ProcessInApplicationTrust(delegate {

        IController controller;

        IControllerFactory factory;

        this.ProcessRequestInit(httpContext, out controller, out factory);

        try

        {

            controller.Execute(this.RequestContext);

        }

        finally

        {

            factory.ReleaseController(controller);

        }

    });

}

На этом этапе уже происходит непосредственно создание контроллера, который представляет объект интерфейса IController. За создание контроллера отвечает объект интерфейса IControllerFactory, представляющий фабрику контроллеров. В качестве класса по умолчанию для фабрики контроллеров выступает класс System.Web.Mvc.DefaultControllerFactory

1. После этого начинается собственно выполнение кода контроллера. После инициализации для запуска метода для обработки запроса контроллер вызывает метод InvokeAction():

public virtual bool InvokeAction(ControllerContext controllerContext, string actionName)

1. Далее привязчик модели (по умолчанию класс System.Web.Mvc.DefaultModelBinder) извлекает данные из запроса, производит из преобразование, форматирование, валидацию и связывает их с определенными параметрами вызываемого метода.
2. При вызове метода в ASP.NET MVC 5 запускается фильтр аутентификации, представляющий объект интерфейса IAuthenticationFilter. Он аутентифицирует пользователя
3. После фильтра аутентификации в ASP.NET MVC 5 запускается фильтр авторизации, представляющий реализацию интерфейса IAuthorizationFilter. До MVC 5 фильтры аутентификации и авторизации объединялись в один фильтр и срабатывали вместе. Фильтр авторизации управляет доступом пользователя к определенным ресурсам на основе его учетных данных.
4. Перед непосредственным выполнением метода контроллера запускается метод OnActionExecuting фильтра действий. Фильтр действий представляет объект интерфейса IActionFilter. Кроме того, также после выполнения метода контроллера запускается другой метод фильтра действий - метод OnActionExecuted
5. Собственно выполнение метода контроллера. Он выполняет определенную логику и на выходе генерирует результат обработки в виде объекта ActionResult.
6. При обработке результата срабатывает другой фильтр - фильтр результатов - объект интерфейса IResultFilter. Его методOnResultExecuting срабатывает до обработки результата, а метод OnResultExecuted после.
7. Генерация результата представляет создание объекта одного из классов результатов действий - ViewResult, ContentResult, FileResult, RedirectResult и др.
8. На заключительном этапе у каждого объекта ActionResult вызывается метод ExecuteResult, который обрабатывает результат действия. Для объектов ViewResult и PartialViewResult это выражается в поиске необхожимого представления и его рендеринге движком представлений (как правило, движком Razor), который представляет объект интерфейса IViewEngine. Результат обработки в виде html-страницы посылается пользователю.
9. Для других объектов ActionResult (ContentResult, RedirectRefult и др.) происходит простая отправка результата в выходной поток.
10. И в конце пользователь получает результат обработки своего запроса.

Схематично весь процесс конвейера приложения на ASP.NET MVC 5 можно представить так:







