ASP.NET MVC

Инструктор: Максим

Содержание

1.Контроллеры
- Переадресация
- Отправка ошибок и статусных кодов
- Отправка файлов
- Контекст запроса HttpContext
- Куки
- Сессии

1. Контроллеры. Переадресация

Существует два вида переадресации: временная и постоянная. И в зависимости от вида переадресации при ее выполнении сервер посылает браузерам один из двух кодов HTTP:

- статусный код 301 представляет постоянную переадресацию. При данном типе переадресации предполагается, что запрашиваемый документ окончательно перемещен в другое место. После получения данного статусного кода браузер может автоматически настраивать запросы на новый ресурс, даже если старый ресурс со временем перестанет применять переадресацию. Поэтому данный способ использовать нежелательно
- статусный код 302 представляет временную переадресацию. При временной переадресации считается, что запрашиваемый документ временно перемещен на другую страницу

В обоих случаях для переадресации будет использоваться объект **RedirectResult**, однако метод, возвращающий данный объект, будет отличаться

1. Контроллеры. Переадресация

Для временной переадресации применяется метод **Redirect**:

```
public RedirectResult SomeMethod()
{
    return Redirect("/My/Index");
}
```

Для постоянной переадресации подобным образом используется метод **RedirectPermanent**:

```
public RedirectResult SomeMethod()
{
    return RedirectPermanent("/My/Index");
}
```

1. Контроллеры. Переадресация

При этом необязательно возвращать из метода объект RedirectResult. Нередко возникает ситуация, когда в зависимости от некоторых условий требуется направить пользователя по одному адресу, либо переадресовать на другой ресурс:

```
public ActionResult Iterator(int i)
{
    if (i < 0)
    {
        return Redirect("/My/Index");
    }
    ViewBag.I = ++i;
    return View();
}
```

1. Контроллеры. Переадресация

Если в качестве параметра будет передано число меньше 0, то произойдет редирект на My/Index. В остальных случаях пользователю будет возвращаться представление

Еще один класс для создания переадресации – **RedirectToRouteResult** – позволяет выполнить более детальную настройку перенаправлений. Он возвращается двумя методами: **RedirectToRoute** и **RedirectToAction**:

Метод RedirectToRoute позволяет произвести перенаправление по определенному маршруту внутри домена:

```
public RedirectToRouteResult SomeMethod()
{
    return RedirectToRoute(new { controller = "Home",
action = "Index" });
}
```

1. Контроллеры. Переадресация

Метод **RedirectToAction** позволяет перейти к определенному действию определенного контроллера. Он также позволяет задать передаваемые параметры:

```
public RedirectToRouteResult SomeMethod()
{
    return RedirectToAction("Square", "My", new { a = 10, h
= 12 });
}
```

Методы RedirectToRoute/RedirectToAction представлют временную переадресацию. Но они имеют свои двойники для создания постоянной переадресации: RedirectToRoutePermanent/RedirectToActionPermanent. Их действие аналогично, разница лишь в том, что они отправляют браузеру статусный код 301. Однако методы RedirectToRoutePermanent и RedirectToActionPermanent не рекомендуется использовать, а если и использовать, то с осторожностью. Так как неправильно настроенная постоянная переадресация может ухудшить позиции в поисковиках или способствовать полному выпадению сайта из поиска

1. Контроллеры. Отправка ошибок и статусных кодов

Иногда возникает необходимость отправить сообщения об ошибках при доступе к тому или иному ресурсу. Обычно, если ресурс недоступен, MVC-фреймворк автоматически отреагирует на эту ситуацию, отправив соответствующий статусный код. Но в некоторых ситуациях нужно более тонко реагировать на полученный запрос. Например, есть контент, к которому установлены возрастные ограничения. Смотрим введенный возраст, и если он попадает под ограничение, можем выслать статусный код ошибки:

```
public ActionResult Check(int age)
{
    if (age < 21)
    {
        return new HttpStatusCodeResult(404);
    }
    return View();
}
```

Подобным образом можно послать браузеру любой другой статусный код

1. Контроллеры. Отправка ошибок и статусных кодов

В качестве альтернативы также можно возвращать объект HttpNotFoundResult с помощью метода **HttpNotFound**

```
public ActionResult Check(int age)
{
    if (age < 21)
    {
        return HttpNotFound();
    }
    return View();
}
```

И еще один класс, предназначенный для отправки статусных кодов – класс HttpUnauthorizedResult. Он извещает пользователя, что тот не имеет права доступа к ресурсу, отправляя браузеру статусный код 401:

```
public ActionResult Check(int age)
{
    if (age < 21)
    {
        return new HttpUnauthorizedResult();
    }
    return View();
}
```

1. Контроллеры. Отправка файлов

Для отправки клиенту файлов предназначен **FileResult**. Однако так как это абстрактный класс, то фактически будем иметь дело с его наследниками:

- **FilePathResult** – представляет простую отправку файла напрямую с сервера
- **FileContentResult** – отправляет клиенту массив байтов, считанный из файла
- **FileStreamResult** – данный класс создает поток – объект System.IO.Stream, с помощью которого считывает и отправляет файл клиенту

Во всех трех случаях для отправки файлов применяется метод **File**, который и возвращает объект FileResult. Только в зависимости от выбранного способа используется соответствующая перегруженная версия этого метода

1. Контроллеры. Отправка файлов

Чтобы отправить файл из файловой системы (то есть использование объекта FilePathResult), надо указать в методе File три параметра: путь к файлу на стороне сервера, тип содержимого и имя файла для принимающей стороны (имя файла необязательно, и можно обойтись в принципе только двумя параметрами):

```
public FileResult GetFile()
{
    // Путь к файлу
    string path = Server.MapPath("~/Files/PDFIcon.pdf");
    // Тип файла - content-type
    string type = "application/pdf";
    // Имя файла - необязательно
    string name = "PDFIcon.pdf";
    return File(path, type, name);
}
```

1. Контроллеры. Отправка файлов

Предполагается, что в проекте есть папка Files, в которой лежит файл PDFIcon.pdf. Метод **Server.MapPath** позволяет построить полный путь к ресурсу из каталога в проекте. Но также можно использовать и абсолютные пути, обращаясь к любому файлу в файловой системе, например, string path = @"C:\Book\PDFIcon.pdf";

И, при обращении, например, по пути My/GetFile будет предложено сохранить данный файл на локальном компьютере

Похожим образом работает и класс FileContentResult, только вместо имени файла в методе File указывается массив байтов, в который был считан файл:

```
public FileResult GetBytes()
{
    string path = Server.MapPath("~/Files/PDFIcon.pdf");
    byte[] mas = System.IO.File.ReadAllBytes(path);
    string type = "application/pdf";
    string name = "PDFIcon.pdf";
    return File(mas, type, name); }
```

1. Контроллеры. Отправка файлов

Если хотим возвратить объект FileStreamResult, то в качестве первого аргумента в методе File идет объект Stream для отправляемого файла:

```
public FileResult GetStream()
{
    string path = Server.MapPath("~/Files/PDFIcon.pdf");
    // Объект Stream
    FileStream fs = new FileStream(path, FileMode.Open);
    string type = "application/pdf";
    string name = "PDFIcon.pdf";
    return File(fs, type, name);
}
```

1. Контроллеры. Контекст запроса HttpContext

Нередко для обработки запроса требуется информация о контексте запроса: какой у пользователя браузер, ip-адрес, с какой страницы или сайта пользователь попал на контроллер. И ASP.NET MVC позволяет получить всю эту информацию, используя объект **HttpContext**

Хотя в контроллере также можно обратиться к объекту ControllerContext, который имеет свойство HttpContext и по сути предоставляет доступ к той же функциональности и информации. Но в то же время между ними есть некоторые различия. Объект HttpContext описывает данные конкретного http-запроса, который обрабатывается приложением. А ControllerContext описывает данные http-запроса непосредственно по отношению к данному контроллеру

1. Контроллеры. Контекст запроса HttpContext

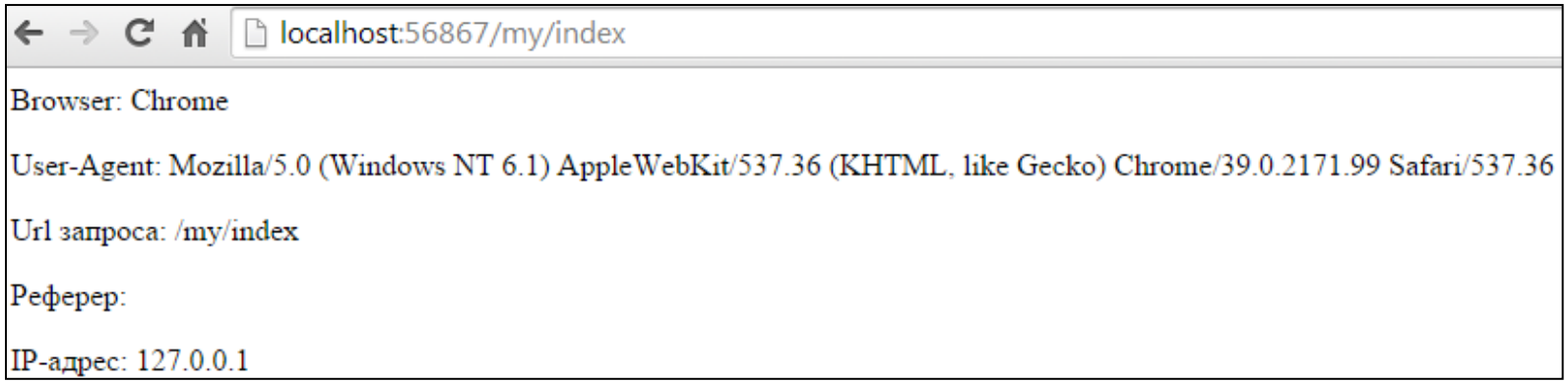
Вся информация о контексте запроса доступна через свойства объекта HttpContext. Так, все данные запроса содержатся в свойстве Request. **HttpContext.Request** представляет объект класса, унаследованного от HttpRequestBase, и поэтому содержит все его свойства:

- **HttpContext.Request.Browser** – получение браузера пользователя
- **HttpContext.Request.UserAgent** – иногда просто браузера недостаточно, тогда можно обратиться к агенту пользователя
- **HttpContext.Request.RawUrl** – получение url запроса
- **HttpContext.Request.UserHostAddress** – получение IP-адреса пользователя
- **HttpContext.Request.UrlReferrer** == null ? "" : HttpContext.Request.UrlReferrer.AbsoluteUri – получение реферера. Так как реферер может быть не определен, то сначала смотрим, не равен ли он null

1. Контроллеры. Контекст запроса HttpContext

Например:

```
public string Index()
{
    string browser = HttpContext.Request.Browser.Browser;
    string userAgent = HttpContext.Request.UserAgent;
    string url = HttpContext.Request.RawUrl;
    string ip = HttpContext.Request.UserHostAddress;
    string referrer = HttpContext.Request.UrlReferrer ==
null ? "" : HttpContext.Request.UrlReferrer.AbsoluteUri;
    return "<p>Browser: " + browser + "</p><p>User-Agent: "
+ userAgent + "</p><p>Url запроса: " + url +
 "</p><p>Реферер: " + referrer + "</p><p>IP-адрес: " + ip
+ "</p>";
}
```



1. Контроллеры. Контекст запроса HttpContext. Отправка ответа

Если HttpContext.Request содержит информацию о запросе, то свойство **HttpContext.Response** управляет ответом. Оно представляет объект HttpResponse, который передает на сторону клиента некоторые значения: куки, служебные заголовки, сам ответ в виде кода html. Например, установим кодировку ответа:

```
HttpContext.Response.Charset = "iso-8859-2";
```

Методы объекта HttpResponse позволяют управлять ответом. Например, метод AddHeader позволяет добавить к ответу дополнительный заголовок

Кроме того, необязательно вызывать метод View в действия контроллера, чтобы отправить клиенту ответ запроса. Можно воспользоваться методом HttpContext.Response.Write:

```
public void ContextData()
{
    HttpContext.Response.Write("<h1>Hello World</h1>");
}
```

1. Контроллеры. Контекст запроса HttpContext. Определение пользователя

Также объект HttpContext содержит информацию о пользователе в свойстве **HttpContext.User**:

```
// Определяем, принадлежит ли пользователь к администраторам
bool isAdmin = HttpContext.User.IsInRole("admin");
// Аутентифицирован ли пользователь
bool isAuth = HttpContext.User.Identity.IsAuthenticated;
// Логин авторизованного пользователя
string login = HttpContext.User.Identity.Name;
```

1. Контроллеры. Куки

Чтобы получить куки, надо воспользоваться свойством HttpContext.Request.Cookies:

```
string id = HttpContext.Request.Cookies["id"].Value;
```

В данном случае, если установлена на стороне клиента куки "id", то получим ее значение

Однако прежде чем получать значения куки, их естественно надо установить. Для установки значения куки можно использовать свойство HttpContext.Response:

```
HttpContext.Response.Cookies["id"].Value = "ca-4353w";
```

1. Контроллеры. Сессии

Сессии также, как и куки, используются для хранения некоторых данных, которые можно получить в любом месте приложения. Для работы с ними используется объект Session, запись данных в сессию:

```
public ActionResult Index()
{
    Session["name"] = "Tom";
    return View();
}
```

Получение данных из сессии:

```
public string GetName()
{
    string val = Session["name"];
    return val;
}
```

Обратите внимание, если необходимо удалить значение сессии для ключа name, можно просто присвоить значение null: Session["name"] = null;

Источники

http://www.asp.net
http://metanit.com
http://professorweb.ru
http://msdn.microsoft.com

Спасибо за внимание!