Распределенные информационно-аналитические системы

Практическое занятие № 14. «URL Rewriting»

Профессор кафедры КБ-2: д.т.н. Шатовкин Р.Р.

Учебные вопросы:

1. **Введение в URL Rewriting.**
2. **Правила IIS для URL Rewriting.**
3. **Применение правил Apache для URL Rewriting.**
4. **Создание правил URL Rewriting.**

1. Введение в URL Rewriting

Функциональность **URL Rewriting** позволяет контролировать доступ к определенным URL-адресам в приложении. В частности, мы можем выполнить переопределение адресов, которые используются для доступа к ресурсам приложения. Например, **URL Rewriting** позволяет решить такие стандартные проблемы, как перенаправление с домена с www на домен без www и наоборот или перенаправление с протокола http на https.

URL Rewriting реализуется до того, как запрос попадет в систему маршрутизации, и начнется его непосредственное выполнение в конвейере **MVC**. Запрошенный адрес изначально может отсутствовать в приложении, однако **URL Rewriting** может изменить этот адрес на любой приемлемый.

Для подключения компонента **URL Rewriting** используется метод расширения **UseRewriter()**, который в качестве параметра принимает объект **RewriteOptions**, задающий все правила переопределения адресов **URL**:

```
using Microsoft.AspNetCore.Rewrite; // Пакет с middleware URL Rewriting

var builder = WebApplication.CreateBuilder();

var app = builder.Build();

// подключаем URL Rewriting
var options = new RewriteOptions();
app.UseRewriter(options);

app.MapGet("/home", async context => await context.Response.WriteAsync("Hello World!"));

app.Run();
```

Правда, в данном случае для **RouteOptions** пока еще не определено никаких правил переопределения **URL**, которые мы можем задать с помощью **специальных методов**:

AddRedirect(): выполняет переадресацию с отправкой статусного кода HTTP 301.

AddRewrite(): подменяет один адрес другим.

AddRedirectToWww(): выполняет переадресацию на поддомен WWW.

AddRedirectToWwwPermanent(): выполняет переадресацию на поддомен WWW с отправкой статусного кода HTTP 301 (постоянная переадресация).

AddRedirectToNonWww(): выполняет переадресацию с поддомена WWW на основной домен.

AddRedirectToNonWwwPermanent(): выполняет переадресацию с поддомена WWW на основной домен с отправкой статусного кода HTTP 301 (постоянная переадресация).

AddRedirectToHttps(): выполняет переадресацию на протокол HTTPS.

AddRedirectToHttpsPermanent(): выполняет переадресацию на протокол HTTPS с отправкой статусного кода HTTP 301 (постоянная переадресация).

AddIISUrlRewrite(): в качестве источника правил для переопределения URL использует правила для IIS.

AddApacheModRewrite(): в качестве источника правил для переопределения URL использует правила для Apache.

AddRedirect

Метод **AddRedirect()** реализован как метод расширения для типа **RewriteOptions** и имеет две формы:

```
1  public static RewriteOptions AddRedirect (this RewriteOptions options, string regex, string replacement);
2  public static RewriteOptions AddRedirect (this RewriteOptions options, string regex, string replacement, int statusCode);
```

В качестве параметра **regex** метод принимает регулярное выражение, которому должен соответствовать входящий адрес **URL**. В качестве адреса в метод **AddRedirect** передается та часть **URL**, которая образуется с помощью объединения значений **Request.Path** и **Request.QueryString**. То есть, если полный запрошенный адрес http://localhost:1234/home/index?id=4, то в метод **AddRedirect** передается home/index?id=4.

Параметр **replacement** представляет выражение, которое указывает, по какому адресу нужно выполнять переадресацию.

Параметр **statusCode** устанавливает отправляемый статусный код.

Например, нам надо, чтобы с адресов с конечным слешем (например, localhost/home/) шло перенаправление на тот же адрес только без слеша (например, localhost/home):

```
using Microsoft.AspNetCore.Rewrite; // Пакет с middleware URL Rewriting

var builder = WebApplication.CreateBuilder();

var app = builder.Build();

// подключаем URL Rewriting
var options = new RewriteOptions().AddRedirect("(.*)/$", "$1");
app.UseRewriter(options);

app.MapGet("/home", async context => await context.Response.WriteAsync("Hello World!"));

app.Run();
```

Регулярное выражение **"(.*)/$"** указывает на любой адрес, который завершается слешем. Второй параметр указывает, что в качестве адреса для перенаправления будет использоваться та часть исходного **URL**, которая идет до слеша: **(.*)**. То есть **"$1"** указывает на первую группу в регулярном выражении **"(.*)/$"**.

То есть в данном случае удаляется концевой слеш (например, перенаправляется с "localhost:1234/home/" на "localhost:1234/home").

Рассмотрим другую ситуацию. Например, мы хотим перенаправлять с адреса home/ на home/index:

```
using Microsoft.AspNetCore.Rewrite; // Пакет с middleware URL Rewriting

var builder = WebApplication.CreateBuilder();

var app = builder.Build();

// подключаем URL Rewriting
var options = new RewriteOptions()
                    .AddRedirect("home[/]?$", "home/index"); // переадресация с home на home/index
app.UseRewriter(options);

app.MapGet("/", async context => await context.Response.WriteAsync("Hello World!"));
app.MapGet("/home", async context => await context.Response.WriteAsync("Home Page!"));
app.MapGet("/home/index", async context => await context.Response.WriteAsync("Home Index Page!"));

app.Run();
```

Для примера с помощью метода **app.MapGet** заданы тестовые маршруты, в итоге при обращении по адресу "home" произойдет переадресация на адрес "home/index", и мы увидим в браузере строку "Home Index Page!".

При этом можно задать последовательно сразу несколько правил:

```
var options = new RewriteOptions()
                .AddRedirect("home[/]?$", "home/index") // переадресация с home на home/index
                .AddRedirect("(.*)/$", "$1");           // удаление концевого слеша
```

AddRewrite

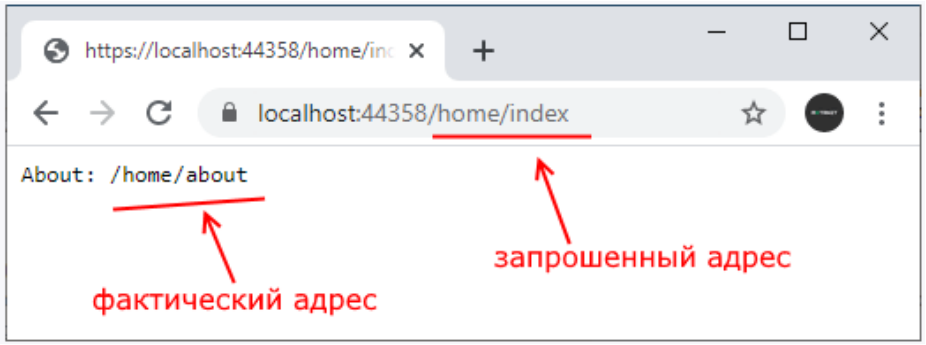
Метод **AddRewrite()** подменяет входящий адрес другим. Первый параметр метода указывает на регулярное выражение, которому должен соответствовать адрес. Второй параметр указывает, на какой адрес надо подменить входящий. Третий параметр – булевое значение устанавливает, надо ли прекратить применение остальных правил, если адрес соответствует выражению из первого параметра. Например:

```
using Microsoft.AspNetCore.Rewrite; // Пакет с middleware URL Rewriting

var builder = WebApplication.CreateBuilder();

var app = builder.Build();

// подключаем URL Rewriting
var options = new RewriteOptions()
            .AddRedirect("(.*)/$", "$1")
            .AddRewrite("home/index", "home/about", skipRemainingRules: false);
app.UseRewriter(options);

app.MapGet("/", async context => await context.Response.WriteAsync("Hello World!"));
app.MapGet("/home/about", async context =>
        await context.Response.WriteAsync($"About: {context.Request.Path}"));
app.MapGet("/home/index", async context =>
        await context.Response.WriteAsync("Home Index Page!"));

app.Run();
```

Правило

```
AddRewrite("home/index", "home/about", skipRemainingRules: false);
```

указывает, что при запросе "home/index" в реальности запрос будет сопоставляться с маршрутом "home/about".



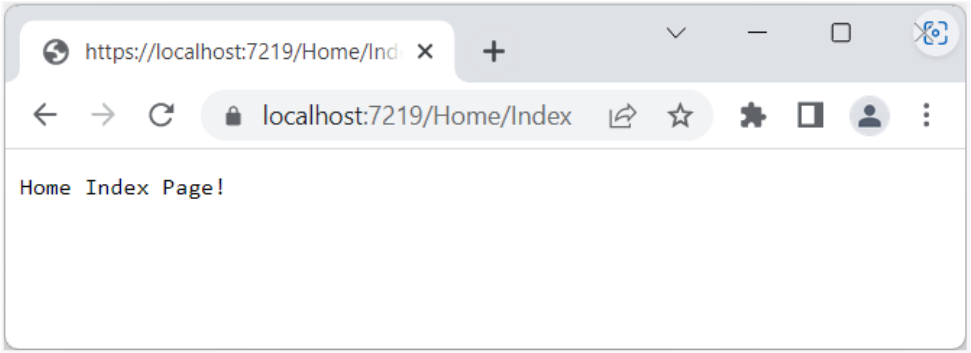
При этом переадресации как таковой нет, статусный код 301/302 не отправляется клиенту.

Регистрозависимость

Стоит отметить, что по умолчанию шаблоны в методах **Addredirect**/**AddRewrite регистрозависимы**. Что это значит? Возьмем в предыдущем примере следующее правило:

```
AddRewrite("home/index", "home/about", skipRemainingRules: false);
```

При запросе "home/index" запрос будет сопоставляться с маршрутом "home/about", однако запрос "Home/Index" по-прежнему будет сопоставляться с запросом "home/index".



Но мы можем выйти из данной ситуации, предваряя шаблон элементом **(?i)**:

```
var options = new RewriteOptions()
            .AddRedirect("(.*)/$", "$1")
            //шаблон регистронезависимый
            .AddRewrite("(?i)home/index", "home/about", skipRemainingRules: false);
```

Элементы регулярных выражений в URL Rewriting

Ключевым элементом, который используется при определении шаблонов адресов, являются группы – набор выражений, которые заключаются в скобки.

Например, в рассмотренном выше примере с удалением концевого слеша применялась одна группа:

```
"(.*)/$"
```

то есть знак точки **"."** означает любой символ, знак звездочки **"*"** означает, что таких символов может быть произвольное количество. И все это объединяется в одну группу – **"(.*)"**. Таким образом, в данном случае группой будет все символы, которые идут до конечного слеша.

При создании паттерна для редиректа или рерайтинга мы можем ссылать на группу по номеру. В примере с концевым слешем определяется одна группа, поэтому мы можем к ней обратиться через **"$1"** – после символа **$** идет номер группы.

Для понимания работы групп при рерайтинге/редиректе рассмотрим несколько примеров:

```
var options = new RewriteOptions()
    .AddRedirect("(.*)/$", "$1")
    .AddRewrite(@"home/index/(\d+)", "home/about?id=$1", skipRemainingRules: false);
```

В данном случае если адрес соответствует выражению **"home/index/(\d+)"** (например, **"home/index/3"**), то фактически происходит обращение по адресу "home/about?id=$1" – **$1** здесь также указывает на первую группу в регулярном выражении – **(\d+)**.

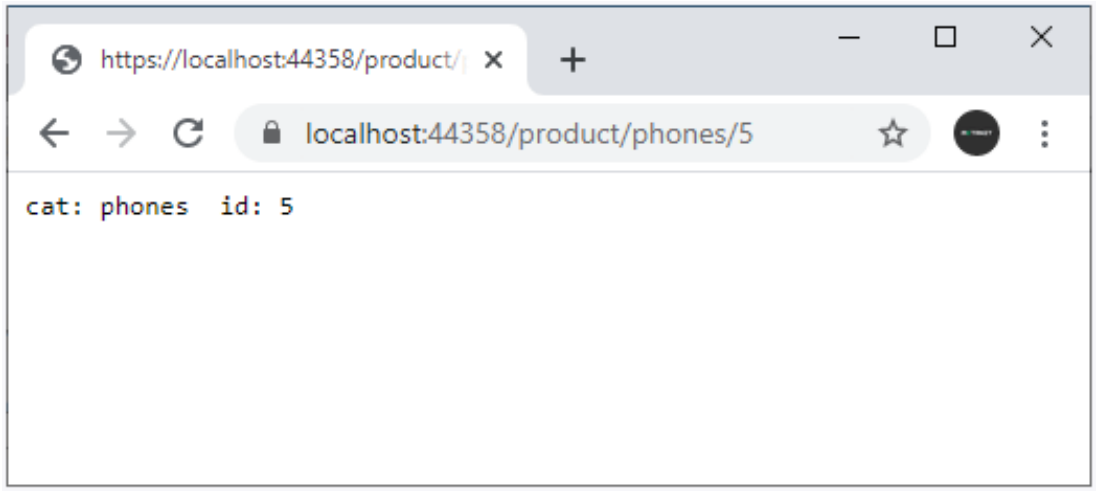
Другой пример. Определим следующее правило **url rewriting**:

```
using Microsoft.AspNetCore.Rewrite; // Пакет с middleware URL Rewriting

var builder = WebApplication.CreateBuilder();

var app = builder.Build();

// подключаем URL Rewriting
var options = new RewriteOptions()
            .AddRewrite(@"product/(\w+)/(\d+)",
                "home/products?cat=$1&id=$2",
                skipRemainingRules: false);
app.UseRewriter(options);

app.MapGet("/", async context =>
{
    await context.Response.WriteAsync("Hello World!");
});
app.MapGet("/home/products", async context =>
{
    await context.Response.WriteAsync($"cat: {context.Request.Query["cat"]} id: {context.Request.Query["id"]}");
});
app.Run();
```

В данном случае используется следующее правило:

```
AddRewrite(@"product/(\w+)/(\d+)", "home/products?cat=$1&id=$2",
                            skipRemainingRules: false);
```

Это правило будет транслировать любой запрос типа **"product/tablet/23"** в запрос типа **"home/products/cat=tablet?id=23"**. То есть теперь у нас две группы: **(\w+)** и **(\d+)**. Соответственно мы к ним можем обратиться через **$1** и **$2**.



В заключении стоит отметить, что использование **url rewriting** увеличивает накладные расходы на обработку запроса. Поэтому стоит по возможности избегать сложных комплексных правил переопределения строки запроса.

Кроме того, **Rewriting Middleware** не покрывает всех возможностей нативных модулей в **IIS**, **Apache**, **Nginx**, которые обеспечивают **URL-rewriting**. Также, нативные модули выше упомянутых веб-серверов демонстрируют большую производительность.

2. Правила IIS для URL Rewriting

В более ранних технологиях на платформе **ASP.NET** (например, в ASP.NET MVC 5) правила для **URL Rewriting** задавались в основном для **IIS** с помощью файла конфигурации **web.config**. И в **ASP.NET Core** мы также можем использовать все эти правила с помощью специального метода **AddIISUrlRewrite()**.

Итак, добавим в корень проекта новый xml-файл urlrewrite.xml:

```
<rewrite>
  <rules>
    <rule name="Redirect from home to home/index">
      <match url = "^home$"/>
      <conditions>
          <add input="{REQUEST_URI}" pattern="home" />
      </conditions>
      <action type="Redirect" url ="home/index" />
    </rule>
  </rules>
</rewrite>
```

Здесь определено одно правило, которое выполняет переадресацию с адреса "/home" на адрес "/home/index". Теперь применим это правило в классе **Startup**:

```
using Microsoft.AspNetCore.Rewrite; // Пакет с middleware URL Rewriting

var builder = WebApplication.CreateBuilder();

var app = builder.Build();

IHostEnvironment? env = app.Services.GetService<IHostEnvironment>();
if(env is not null)
{
    var options = new RewriteOptions()
                    .AddIISUrlRewrite(env.ContentRootFileProvider, "urlrewrite.xml");
    app.UseRewriter(options);
}

app.MapGet("/", async context => await context.Response.WriteAsync("Hello World!"));
app.MapGet("/home", async context =>
        await context.Response.WriteAsync("Home Page!"));
app.MapGet("/home/index", async context =>
        await context.Response.WriteAsync("Home Index Page!"));

app.Run();
```

В качестве первого параметра в **AddIISUrlRewrite** передается провайдер файлов. В данном случае используем встроенный провайдер из добавляемого по умолчанию сервиса **IHostEnvironment**.

Второй параметр представляет путь к файлу.

Также загружать файл конфигурации следующим образом:

```
using System.IO;

using (StreamReader iisReader = File.OpenText("urlrewrite.xml"))
{
    var options = new RewriteOptions().AddIISUrlRewrite(iisReader);
    app.UseRewriter(options);
}
```

Все правила определяются в элементе **<rules>**. Каждое правило, представленное элементом **<rule>**, состоит из **трех частей**:

Pattern – выражение, которому должна соответствовать строка запроса и которое задается в элементе **<match>**.

Conditions – различные дополнительные условия, которым должен соответствовать URL-адрес. Например, значения HTTP-заголовков, пути к файлам и т.д.

Action – действие, которое должно выполняться, если строка **URL** соответствует регулярному выражению в **Pattern** и условиям **Conditions**.

Мы можем использовать несколько правил одновременно, но их выполнение не всегда обязательно. Поэтому у каждого элемента **rule** определен атрибут **StopProcessing**. Если имеет значение **true**, то после выполнения действия в элементе **<action>** адрес **URL**, создаваемый данным правилом, передается в конвейер обработки запроса, а другие правила не будут обрабатываться.

Переопределение URL имеет следующий порядок действий:

1. Строка запроса сравнивается с выражением в элементе **match**. Если обнаружится, что запрошенный адрес не соответствует выражению, то модуль **URL Rewrite Module** прекращает обрабатывать текущее правило и переходит к следующему (если задано несколько правил).

2. Если строка запроса соответствует выражению в элементе **match** и при этом не задано никаких дополнительных условий с помощью элемента **<conditions>**, то **URL Rewrite Module** выполняет действие, которое определено в правиле с помощью элемента **<action>**.

3. Если строка запроса соответствует выражению в элементе **match** и также определены дополнительные условия, то **URL Rewrite Module** проверяет эти условия. И если **URL** соответствует этим условиям, то выполняется действие **action**.

Определение условий

Условия, задаваемые элементом **<conditions>**, определяют дополнительную логику оценки **URL** на соответствие правилу. Каждое отдельное условие задается с помощью элемента **<add >** и настраивается с помощью **следующих атрибутов**:

input: определяет объект, который будет использоваться условием для оценки. В частности, в примере выше используется **input="{REQUEST_URI}"**, где **"REQUEST_URI"** представляет переменную сервера, хранящую запрошенный адрес **url**. Тут также могут использоваться и другие переменные сервера.

pattern: определяет регулярное выражение, которому должен соответствовать объект.

matchType: принимает следующие значения:

Pattern: в этом случае объект (в данном случае адрес URL) сопоставляется с выражением в атрибуте **pattern**. При других значениях атрибут **pattern** не учитывается.

IsFile: определяет, является ли объект (адрес URL) файлом в файловой системе.

IsDirectory: определяет, является ли объект (адрес URL) каталогом в файловой системе.

ignoreCase: указывает, надо ли игнорировать регистр адреса URL. По умолчанию равно **true**, поэтому регистр не учитывается.

negate: если равно **true**, то правило применяется, если условие НЕ учитывается. По умолчанию равно f**alse**.

Определение действий

Если выражение и условия, определяемые правилом, соответствуют объекту (например, адресу URL), то выполняется определенное действие, заданное элементом **<action>**. Действия могут быть нескольких типов. Тип задается с помощью атрибута **type**, который принимает следующие **значения**:

Rewrite: заменяет текущую строку запроса URL другой строкой.

Redirect: выполняет редирект, посылая клиенту статусный код 3xx.

CustomResponse: отправляет клиенту определенный статусный код, а также может отправлять специфическое сообщение.

AbortRequest: сбрасывает подключение для текущего клиента.

Другие атрибуты элемента **action**:

url: строка, которая будет заменять текущую строку запроса URL.

appendQueryString: определяет, должна ли сохраняться та часть строки запроса, которая идет после названия домена и порта. По умолчанию имеет значение **true**, что значит, что строка запроса со всеми параметрами за исключением названия домена будет сохраняться.

redirectType: статусный код переадресации при использовании типа **Redirect** (301 – Permanent, 302 – Found, 303 – See other, 307 – Temporary).

statusCode: определяет статусный код в качестве ответа клиенту при использовании типа **CustomResponse**.

subStatusCode: определяет вспомогательный статусный код при использовании типа **CustomResponse**.

statusReason: определяет сообщение, отправляемое клиенту вместе со статусным кодом при использовании типа **CustomResponse**.

statusDescription: определяет сообщение, отправляемое клиенту в теле ответа при использовании типа **CustomResponse**.

Использование переменных сервера

При изменении **URL** мы можем использовать следующие **переменные сервера**:

QUERY_STRING: параметры запроса.

HTTP_HOST: домен.

SERVER_PORT: номер порта.

SERVER_PORT_SECURE и HTTPS: указывают, использует ли клиент защищенное подключение.

REQUEST_URI: полная строка запроса.

URL представляется в следующем виде: http(s)://<host>:<port>/<path>?<querystring>. Допустим, пользователь обращается к URL http://www.somesite.com/home/index?id=2&name=3. Тогда **IIS** сегментирует ее следующим образом:

path: представляет сегмент home/index. Эта часть затем сравнивается правилом с выражением, определенным в элементе **<match>**.

QUERY_STRING: в данном случае сегмент параметров id=2&name=3.

HTTP_HOST: сегмент www.somesite.com.

SERVER_PORT: если номер порта не указан, то по умолчанию равен **80**.

SERVER_PORT_SECURE равен **0**, а **HTTPS** содержит **OFF**.

REQUEST_URI: сегмент home/index?id=2&name=3.

При создании условий для правил мы можем ссылаться на эти переменные через выражение вида **"{НАЗВАНИЕ_ПЕРЕМЕННОЙ}"**. Например, нам нужно условие, согласно которому в строке параметров должен быть числовой параметр **id**:

```
1 <add input="{QUERY_STRING}" pattern="id=([0-9]+)" />
```

Кроме того, нам доступны заголовки **HTTP**-запроса, например, строку юзер-агента мы можем получить с помощью выражения **"{HTTP_USER_AGENT}"**.

При использовании заголовков запроса надо учитывать, что все дефисы в названии заголовков (например, **User-Agent**) заменяются символами подчеркивания. Все строчные буквы заменяются заглавными, а к названию переменных добавляется префикс **"HTTP_"**. Как например, из заголовка **User-Agent** создается переменная **HTTP_USER_AGENT**.

Обратные ссылки

Обратные ссылки представляют отдельные сегменты выражений, которые используются в условиях. Например:

```
<rewrite>
  <rules>
    <rule name="Redirect">
      <match url = "(.*)"/>
      <conditions>
        <add input="{REQUEST_URI}" pattern="([a-z]+)/([a-z]+)/([0-9]+)" matchType="Pattern" />
      </conditions>
      <action type="Redirect" url ="{C:1}/{C:3}" />
    </rule>
  </rules>
</rewrite>
```

Все обратные ссылки представляют выражения типа **{C:N}**, где **N** – число от 0 до 9. При этом значение **{C:0}** представляет всю попадающую под условие строку.

То есть из строки запроса "home/index/2" генерировались бы три обратных ссылки C:1 = "home", C:2 = "index" и C:3 = "2". И в соответствии с элементом **action** шла бы переадресация на адрес "home/2" (то есть "{C:1}/{C:3}").

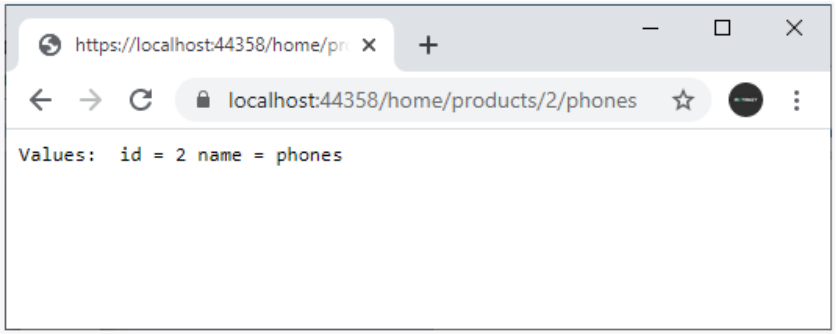
Кроме условий для создания обратных ссылок могут применяться выражения в элементе **match**. Все обратные ссылки из выражения **match** доступны через выражения типа **{R:N}**, где **N** – число от 0 до 9. При этом значение **{R:0}** представляет всю попадающую под условие строку.

Так, рассмотрим другой пример. Допустим, у нас есть правило:

```
<rule name="Rewrite query" stopProcessing="true">
    <match url="^home/products/([0-9]+)/([_0-9a-z-]+)" />
    <action type="Rewrite" url="home/products?id={R:1}&amp;name={R:2}"  />
</rule>
```

Например, при запросе http://localhost:50645/Home/Products/2/phones мы получим следующие сегменты:
{R:0} = "Home/Products/2/phones"
{R:1} = "2"
{R:2} = "phones"
В итоге будет формироваться следующая строка URL: http://localhost:50645/Home/Products?id=2&name=phones.
Для тестирования определим следующее приложение:

```
using Microsoft.AspNetCore.Rewrite; // Пакет с middleware URL Rewriting

var builder = WebApplication.CreateBuilder();

var app = builder.Build();

IHostEnvironment? env = app.Services.GetService<IHostEnvironment>();
if(env is not null)
{
    var options = new RewriteOptions()
                    .AddIISUrlRewrite(env.ContentRootFileProvider, "urlrewrite.xml");
    app.UseRewriter(options);
}

app.MapGet("/", async context => await context.Response.WriteAsync("Hello World!"));
app.MapGet("/home/products", async context =>
    await context.Response.WriteAsync($"Values:  id = {context.Request.Query["id"]} name = {context.Request.Query["name"]}"));

app.Run();
```



Еще один пример – переадресация с домена без www на субдомен www:

```
<rule name="Redirect to WWW" enabled="true" stopProcessing="true">
    <match url=".*" />
    <conditions logicalGrouping="MatchAll">
        <add input="{HTTP_HOST}" pattern=".*" />
    </conditions>
    <action type="Redirect" url="http://www.localhost:50645/{R:0}" />
</rule>
```

Или обратное действие – перенаправление с www на домен без www (для домена localhost):

```
<rule name="Redirect to NonWWW" stopProcessing="true">
    <match url=".*" />
    <conditions logicalGrouping="MatchAll">
          <add input="{HTTP_HOST}" pattern="^localhost" negate="true" />
    </conditions>
    <action type="Redirect" url="http://localhost:50645/{R:0}" />
</rule>
```

3. Применение правил Apache для URL Rewriting

Для написания правил для **URL Rewriting** мы также можем использовать стандартные правила для веб-сервера **Apache**, которые обычно помещаются в файл htaccess. Что может быть особенно полезно, если сайт ранее был написан на PHP, а потом его решили перенести на **ASP.NET Core**. Однако надо заметить, что вряд ли получится взять все правила из htaccess и простым копированием перенести в проект для **ASP.NET Core**, поскольку не все они могут работать или работать так, как на **Apache**. И в случае с каждым правилом лучше, конечно же, тестировать.

Итак, добавим в корень проекта **ASP.NET Core** новый текстовый файл, который назовем rewrite.txt. Определим в этом файле следующее содержимое:

```
# удаление концевого слеша
RewriteRule (.*)/$ $1 [R=301]
# переадресация  home/index на /
RewriteRule "Home/Index$" "/" [NC,R=301]
# подмена адреса типа Home/Products/2/phones на Home/Products?id=2&name=phones
RewriteRule Home/Products/([0-9]+)/([0-9a-z-]+) Home/Products?id=$1&name=$2 [NC]
```

Для переопределения адресов применяется слово **RewriteRule**, которое в качестве параметра принимает выражение, которому должен соответствовать адрес, и ссылку, на которую будет осуществлен переход.

Первое выражение **RewriteRule** получает запрошенный путь, сравнивает его с регулярным выражением, и если сравнение прошло успешно, то преобразует адрес. Последующим **RewriteRule** передается результат предыдущих преобразований.

Регулярные выражения могут иметь группы, ограниченные круглыми скобками. В преобразованном адресе на каждую группу мы можем ссылаться через выражения типа **$1**, где число **1** – это порядковый номер группы. Например, в регулярном выражении **"(.*)/$" $1** передает ту часть пути, которая идет до конечного слеша.

В выражении **"Home/Products/([0-9]+)/([0-9a-z-]+)"** уже две группы, для обращения к которым можно использовать токены **$1** и **$2**, соответственно.

Если надо выполнить переадресацию, то в конце правила используется выражение **[R=301]** или **[R]**. В случае с третьим правилом никакой переадресации нет, будет просто подмена одного адреса на другой.

Кроме того, поскольку при определении правил **Apache** по умолчанию учитывается регистр (то есть **Home** и **home** – это не одно и то же), чтобы регистр не учитывался, также используется флаг **[NC]**.

Более подробно про правила **URL Rewriting**, которые применяются в **Apache**, можно посмотреть в официальной документации.

Для использования этого файла применяется метод **AddApacheModRewrite()**:

```
using Microsoft.AspNetCore.Rewrite; // Пакет с middleware URL Rewriting

var builder = WebApplication.CreateBuilder();

var app = builder.Build();

IHostEnvironment? env = app.Services.GetService<IHostEnvironment>();
if(env is not null)
{
    var options = new RewriteOptions()
                    .AddApacheModRewrite(env.ContentRootFileProvider, "rewrite.txt");
    app.UseRewriter(options);
}

app.MapGet("/", async context => await context.Response.WriteAsync("Hello World!"));
app.MapGet("/home/products", async context =>
    await context.Response.WriteAsync($"Values:  id = {context.Request.Query["id"]} name = {context.Request.Query["name"]}"));

app.Run();
```

В метод **AddApacheModRewrite** передается провайдер файлов, который позволит сформировать путь к файлу правил и также название самого файла. В данном случае провайдер получаем через свойствам **env.ContentRootFileProvider** встроенного сервиса **IHostEnvironment**.

В качестве альтернативы можно использовать другой способ получения конфигурации **Apache** из файла:

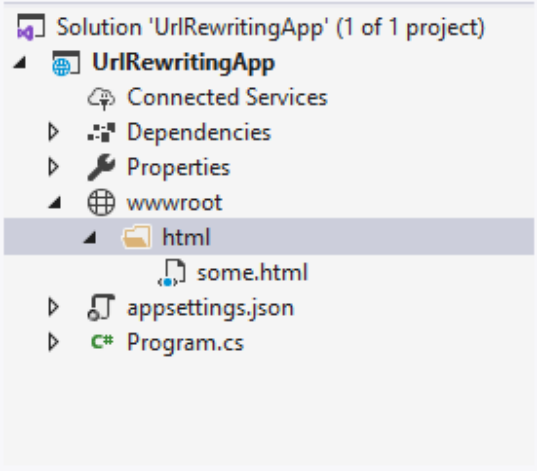
```
using System.IO;

using (StreamReader apacheReader = File.OpenText("rewrite.txt"))
{
    var options = new RewriteOptions().AddApacheModRewrite(apacheReader);
    app.UseRewriter(options);
}
```

4. Создание правил URL Rewriting

Если необходимо использовать какую-то более сложную логику по переопределению строки запроса, то в этом случае мы можем определить свои правила с помощью методов или целых классов.

Например, пусть ранее сайт использовал технологию **php**, но затем мигрировал на asp.net, а все документы **php** были сконвертированы в документы **html**. То есть ранее сайт, к примеру, использовал адрес http://localhost:1234/some.php, то теперь документ перемещен по адресу http://localhost:1234/html/some.html. Рассмотрим на примере создания своих правил, как мы можем решить проблему адресов.

Пусть нужные документы находятся в проекте в папке wwwroot/html:



Допустим, там находится следующая страница some.html:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title></title>
</head>
<body>
    <h2>Hello World!</h2>
</body>
</html>
```

Вначале рассмотрим простой рерайт без переадресации и определим правило в виде отдельного метода. Для этого изменим код в файле Program.cs следующим образом:

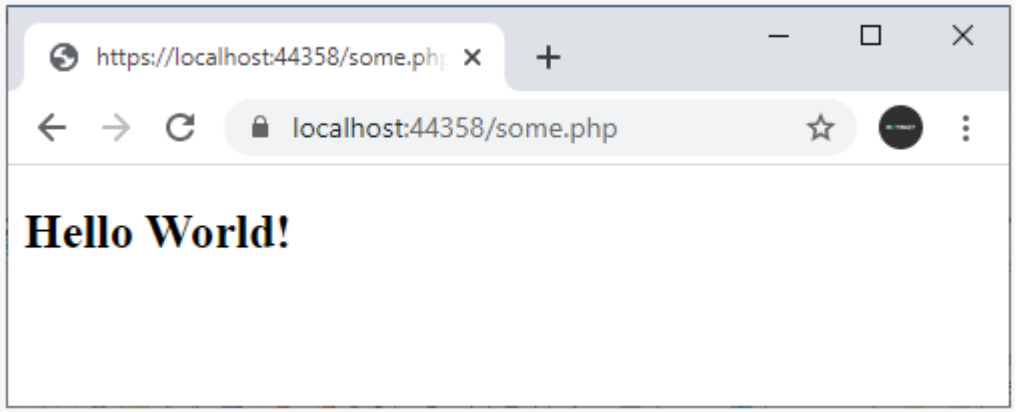
```
using Microsoft.AspNetCore.Rewrite; // Пакет с middleware URL Rewriting

var builder = WebApplication.CreateBuilder();

var app = builder.Build();

var options = new RewriteOptions().Add(RewritePHPRequests);

app.UseRewriter(options);
app.UseStaticFiles();

app.MapGet("/", async context => await context.Response.WriteAsync("Hello World!"));

app.Run();

static void RewritePHPRequests(RewriteContext context)
{
    var path = context.HttpContext.Request.Path;
    var pathValue = path.Value; // запрошенный путь
                                // если запрос к папке html, возвращаем ошибку 404
    if (path.StartsWithSegments(new PathString("/html")))
    {
        context.HttpContext.Response.StatusCode = StatusCodes.Status404NotFound;
        context.Result = RuleResult.EndResponse;
        return;
    }
    // если запрос к файлам с расширением php, переопределяем запрос на папку html
    if (pathValue!=null && pathValue.EndsWith(".php", StringComparison.OrdinalIgnoreCase))
    {
        // отрезаем расширение "php" в запрошенном пути и добавляем "html"
        string proccedPath = "/html" + pathValue.Substring(0, pathValue.Length - 3) + "html";
        context.HttpContext.Request.Path = new PathString(proccedPath);
    }
}
```

Для применения правила у объекта **RewriteOptions** вызывается метод **Add**, в который передается делегат **Action<RewriteContext>**. В данном случае передаем ссылку на метод **RewritePHPRequests**.

В методе **RewritePHPRequests** вначале получаем объекты запроса, ответа и запрошенный путь. Если запрошенный путь уже содержит путь к каталогу html, то отклоняем его, устанавливая в качестве кода ответа статусный код 404. Для завершения выполнения задается значение **context.Result = RuleResult.EndResponse**. Тем самым мы предотвращаем прямой доступ к папке html (допустим, необходимо скрыть путь к документам html).

Если запрошенный адрес заканчивается на ".php", то выпоняем ряд преобразований, получая путь к html-документу в папке webroot/html. И затем устанавливаем новое значение у свойства **request.Path**. Из него последующие компоненты **middleware** будут брать информацию о запрошенном пути и обработать его соответствующим образом.

Запустим приложение на выполнение и обратимся по адресу http://localhost:xxxx/some.php:



В этом случае произойдет обращение к документу webroot/html/some.html.

Рассмотрим другой способ. Допустим, нам надо не просто переопределить адрес внутри приложения, а выполнить постоянную переадресацию, уведомляя браузеры пользователей и поисковики, что адрес документа окончательно изменился. Для этого определим новый класс **RedirectPHPRequests** (хотя можно было бы и в виде метода определить):

```
using System.Text.RegularExpressions;
using Microsoft.AspNetCore.Rewrite;
using Microsoft.Net.Http.Headers;

namespace UrlRewritingApp
{
    public class RedirectPHPRequests : IRule
    {
        private readonly string _extension;
        private readonly PathString _newPath;

        public RedirectPHPRequests(string extension, string newPath)
        {
            if (string.IsNullOrEmpty(extension))
            {
                throw new ArgumentException(nameof(extension));
            }
            if (!Regex.IsMatch(newPath, @"(/[A-Za-z0-9]+)+?"))
            {
                throw new ArgumentException("Запрошенный путь недействителен", nameof(newPath));
            }

            _extension = extension;
            _newPath = new PathString(newPath);
        }

        public void ApplyRule(RewriteContext context)
        {
            var request = context.HttpContext.Request;
            var pathValue = request.Path.Value; // запрошенный путь

            if (request.Path.StartsWithSegments(new PathString(_newPath))) return;

            if (pathValue != null && pathValue.EndsWith(".php", StringComparison.OrdinalIgnoreCase))
            {
                var response = context.HttpContext.Response;

                response.StatusCode = StatusCodes.Status301MovedPermanently;
                context.Result = RuleResult.EndResponse;
                response.Headers[HeaderNames.Location] = _newPath + pathValue.Substring(0, pathValue.Length - 3) + _extension;
            }
        }
    }
}
```

Класс правила должен реализовать интерфейс **IRule**, который определяет метод **ApplyRule**.

В конструкторе получаем расширение, которое стоит использовать вместо php, а также путь к документам в рамках проекта.

В методе **ApplyRule** если вдруг запрошенный адрес начинается с названия каталог, где лежать файлы html, то завершаем выполнение. Так как нет смысла выполнять переадресацию, ведь запрос уже идет к файлам в нужном каталоге. Иначе, если запрошен документ php, извлекаем имя документа и формируем новый путь. Этот путь передается через заголовок **"Location"**. И кроме того, устанавливается статусный код постоянной переадресации 301.

Применим этот класс в файле Program.cs:

```
using UrlRewritingApp;  // пространство имен класса RedirectPHPRequests
using Microsoft.AspNetCore.Rewrite; // Пакет с middleware URL Rewriting

var builder = WebApplication.CreateBuilder();

var app = builder.Build();

var options = new RewriteOptions()
            .Add(new RedirectPHPRequests(extension: "html", newPath: "/html"));

app.UseRewriter(options);
app.UseStaticFiles();

app.MapGet("/", async context => await context.Response.WriteAsync("Hello World!"));

app.Run();
```

Перегруженная версия метода **Add** класса **RewriteOptions** принимает объект **IRule**, в качестве которого в данном случае передается объект **RedirectPHPRequests**.