

Современные платформы прикладной разработки

СОХРАНЕНИЯ СОСТОЯНИЙ

Сохранения состояний

HTTP - это протокол без сохранения состояния. По умолчанию HTTP-запросы представляют собой независимые сообщения, которые не сохраняют пользовательские значения.

TempData

ASP.NET Core предоставляет TempData Razor Pages или контроллера. Это свойство хранит данные до тех пор, **пока они не будут прочитаны** другим запросом. Методы Keep (String) и Peek (string) могут использоваться для проверки данных без удаления в конце запроса.

TempData

```
if (TempData.Peek("Message") != null)
{
    <h3>Message: @TempData.Peek("Message")</h3>
}
```

TempData

```
if (TempData["Message"] != null)
{
    <h3>Message: @TempData["Message"]</h3>
}
TempData.Keep("Message");
```

TempData

- ❑ Полезно для перенаправления, когда данные требуются для более чем одного запроса.
- ❑ Реализуется поставщиками TempData с использованием файлов cookie или состояния сеанса.

Query strings

Ограниченный объем данных может быть передан от одного запроса к другому, добавив их в строку запроса нового запроса.

<https://docs.microsoft.com/ru-ru/aspnet/core/?view=aspnetcore-5.0>

Query strings

Это полезно для постоянного захвата состояния, что позволяет делиться ссылками со встроенным состоянием через электронную почту или социальные сети.

Query strings

Поскольку строки запроса URL являются общедоступными, никогда не используйте строки запроса для конфиденциальных данных.

Скрытые поля формы

Данные могут быть сохранены в скрытых полях формы и отправлены обратно при следующем запросе. Это обычное явление для многостраничных форм. Поскольку клиент потенциально может подделать данные, приложение всегда должно повторно проверять данные, хранящиеся в скрытых полях.

Скрытые поля формы

```
<form asp-action="Edit">  
  <div asp-validation-summary="ModelOnly"  
      class="text-danger"></div>
```

```
  <input type="hidden" asp-for="Id" />
```

.
.
.

```
  <div class="form-group">  
    <input type="submit" value="Save" class="btn  
btn-primary" />  
  </div>  
</form>
```

Кэширование

Кэширование - это эффективный способ хранения и извлечения данных. Приложение может контролировать время жизни кэшированных элементов.

Кэшированные данные не связаны с конкретным запросом, пользователем или сеансом. Не кэшируйте пользовательские данные, которые могут быть получены по запросам других пользователей.

Кэширование (варианты)

- ☐ Кэширование данных в памяти сервера (In-memory или Distributed memory)
- ☐ Кэширование ответа
- ☐ Кэширование запроса

Кэширование (tag-helper)

```
<cache>@DateTime.Now</cache>
```

Первый запрос к странице, содержащей вспомогательную функцию тега, отображает текущую дату. Дополнительные запросы показывают кешированное значение до истечения срока действия кеша (по умолчанию 20 минут) или пока кешированная дата не будет удалена из кеша.

Куки

ХРАНЕНИЕ ДАННЫХ МЕЖУ ЗАПРОСАМИ

Куки

Файлы cookie хранят данные между запросами.

Поскольку файлы cookie отправляются с каждым запросом, их размер должен быть минимальным. В идеале в файле cookie должен храниться только идентификатор с данными, хранящимися в приложении.

Куки

Большинство браузеров ограничивают размер файлов cookie 4096 байтами. Для каждого домена доступно только ограниченное количество файлов cookie.

Куки

Поскольку файлы cookie могут быть изменены, они должны быть проверены приложением.

Файлы cookie могут быть удалены пользователями, а срок их действия истекает на клиентах. Однако файлы cookie, как правило, являются наиболее надежной формой сохранения данных на клиенте.

Куки

Для чтения куки используется коллекция

Request.Cookies

Для записи куки используется коллекция

Response.Cookies

Куки

```
if(!Request.Cookies.ContainsKey("Name"))  
{  
    Response.Cookies.Append("Name", "Bob");  
}
```

Куки

```
string name;  
if (!Request.Cookies.TryGetValue("Name", out name))  
{  
    Response.Cookies.Append("Name", "Bob");  
};
```

Куки

Чтение куки по ключу:

```
name = Request.Cookies["Name"];
```

Куки

Удаление куки:

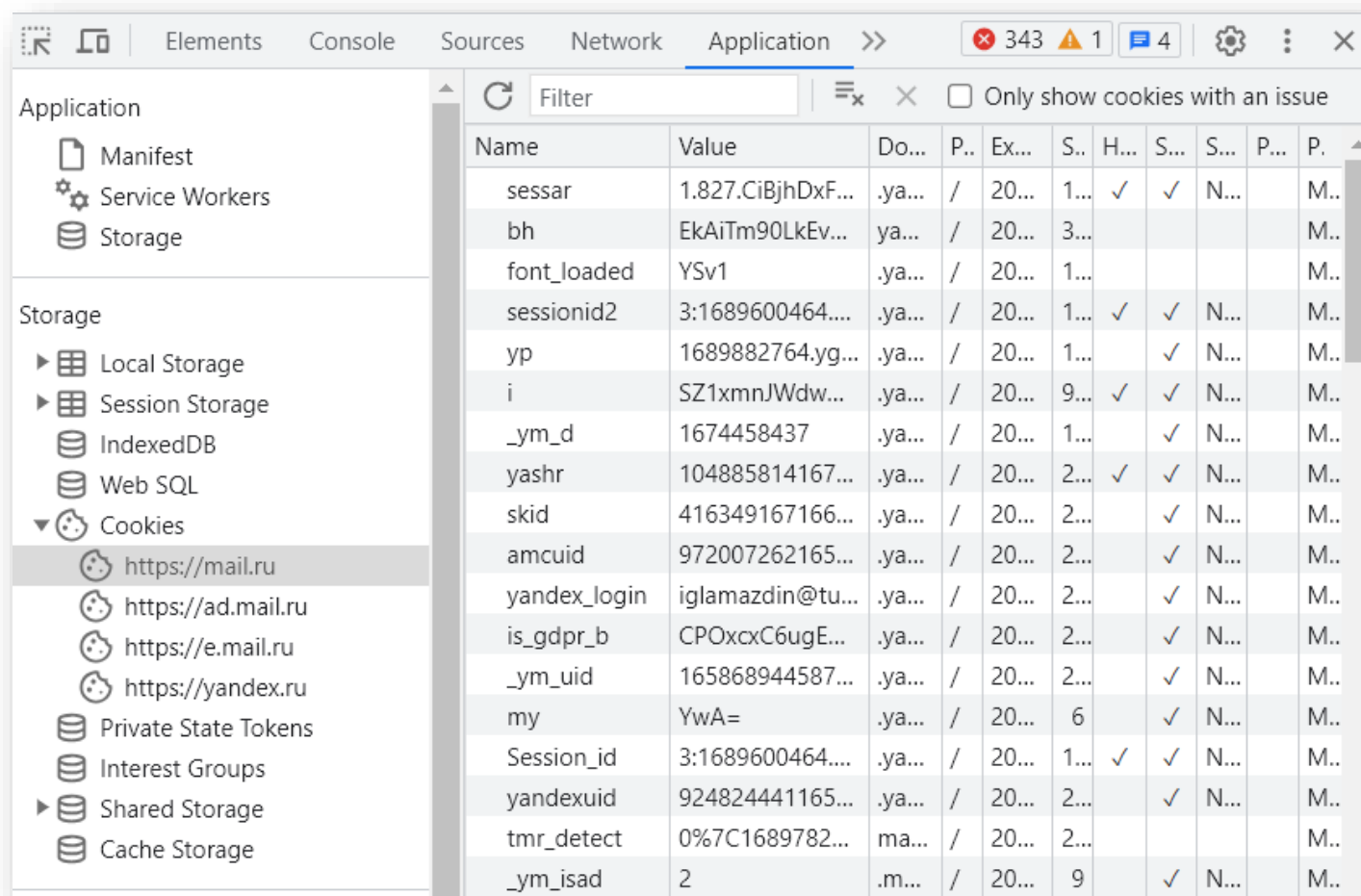
```
Response.Cookies.Delete("Name")
```

Куки

Задание срока жизни куки:

```
var opt = new CookieOptions {  
    Expires= DateTime.Now.AddDays(2) };  
Response.Cookies.Append("Name", "Bob", opt);
```


Куки



Application

- Manifest
- Service Workers
- Storage

Storage

- Local Storage
- Session Storage
- IndexedDB
- Web SQL
- Cookies
 - https://mail.ru
 - https://ad.mail.ru
 - https://e.mail.ru
 - https://yandex.ru
- Private State Tokens
- Interest Groups
- Shared Storage
- Cache Storage

Filter

Only show cookies with an issue

Name	Value	Do...	P..	Ex...	S..	H...	S...	S...	P...	P.
sessar	1.827.CiBjhDxF...	.ya...	/	20...	1...	✓	✓	N...		M..
bh	EkAiTm90LkEv...	ya...	/	20...	3...					M..
font_loaded	YSv1	.ya...	/	20...	1...					M..
sessionid2	3:1689600464....	.ya...	/	20...	1...	✓	✓	N...		M..
yp	1689882764.yg...	.ya...	/	20...	1...		✓	N...		M..
i	SZ1xmnJWdw...	.ya...	/	20...	9...	✓	✓	N...		M..
_ym_d	1674458437	.ya...	/	20...	1...		✓	N...		M..
yashr	104885814167...	.ya...	/	20...	2...	✓	✓	N...		M..
skid	416349167166...	.ya...	/	20...	2...		✓	N...		M..
amcuid	972007262165...	.ya...	/	20...	2...		✓	N...		M..
yandex_login	iglamazdin@tu...	.ya...	/	20...	2...		✓	N...		M..
is_gdpr_b	CPOxcxC6ugE...	.ya...	/	20...	2...		✓	N...		M..
_ym_uid	165868944587...	.ya...	/	20...	2...		✓	N...		M..
my	YwA=	.ya...	/	20...	6		✓	N...		M..
Session_id	3:1689600464....	.ya...	/	20...	1...	✓	✓	N...		M..
yandexuid	924824441165...	.ya...	/	20...	2...		✓	N...		M..
tmr_detect	0%7C1689782...	ma...	/	20...	2...					M..
_ym_isad	2	.m...	/	20...	9		✓	N...		M..

Сессии

ХРАНЕНИЕ ДАННЫХ МЕЖУ ЗАПРОСАМИ

Сессии

Сессия - это сценарий ASP.NET Core для хранения пользовательских данных, пока пользователь просматривает веб-приложение. Состояние сеанса использует хранилище, поддерживаемое приложением, для сохранения данных между запросами от клиента. Данные сеанса поддерживаются кешем. Сайт должен продолжать работать без данных сеанса.

Сессии

Сессия может использоваться для сохранения каких-то временных данных, которые должны быть доступны, пока пользователь работает с приложением, и не требуют постоянного хранения.

Сессии

ASP.NET Core поддерживает состояние сеанса, предоставляя клиенту файл cookie, содержащий идентификатор сеанса.

Идентификатор сеанса cookie:

- ☐ Отправляется в приложение с каждым запросом.
- ☐ Используется приложением для получения данных сеанса.

Сессии

За работу сессии отвечает соответствующее
Middleware.

Сессии

Чтобы включить Middleware сессии, класс Program должен содержать:

- ☐ Любой из кешей памяти IDistributedCache. Реализация IDistributedCache используется как резервное хранилище для сессии.
- ☐ Вызов AddSession.
- ☐ Вызов UseSession.

Файл Program.cs

```
#region Session setup
builder.Services.AddDistributedMemoryCache();
builder.Services.AddSession();
#endregion

. . .

app.UseSession();
```


Сессии

Доступ к состоянию сеанса осуществляется из класса `PageModel` Razor Pages или класса контроллера MVC с помощью `HttpContext.Session`. Это свойство является реализацией `ISession`.

Реализация `ISession` предоставляет несколько методов расширения для установки и получения целочисленных и строковых значений. Методы расширения находятся в пространстве имен `Microsoft.AspNetCore.Http`.

Расширяющие методы

```
using Microsoft.AspNetCore.Http;
```

```
HttpContext.Session.SetString("_name", "Bob");  
HttpContext.Session.SetInt32("_age", 18);
```

```
var name = HttpContext.Session.GetString("_name");  
var age = HttpContext.Session.GetInt32("_age");
```

Сессии

Сложные типы должны сериализоваться пользователем с помощью другого механизма, например JSON.

Сессии

```
using Microsoft.AspNetCore.Http;
using System.Text.Json;

public static void Set<T>(this ISession session, string key, T value)
{
    session.SetString(key, JsonSerializer.Serialize(value));
}
```

Сессии

```
using Microsoft.AspNetCore.Http;  
using System.Text.Json;
```

```
public static T Get<T>(this ISession session, string key)  
{  
    var value = session.GetString(key);  
    return value == null  
        ? default  
        : JsonSerializer.Deserialize<T>(value);  
}
```

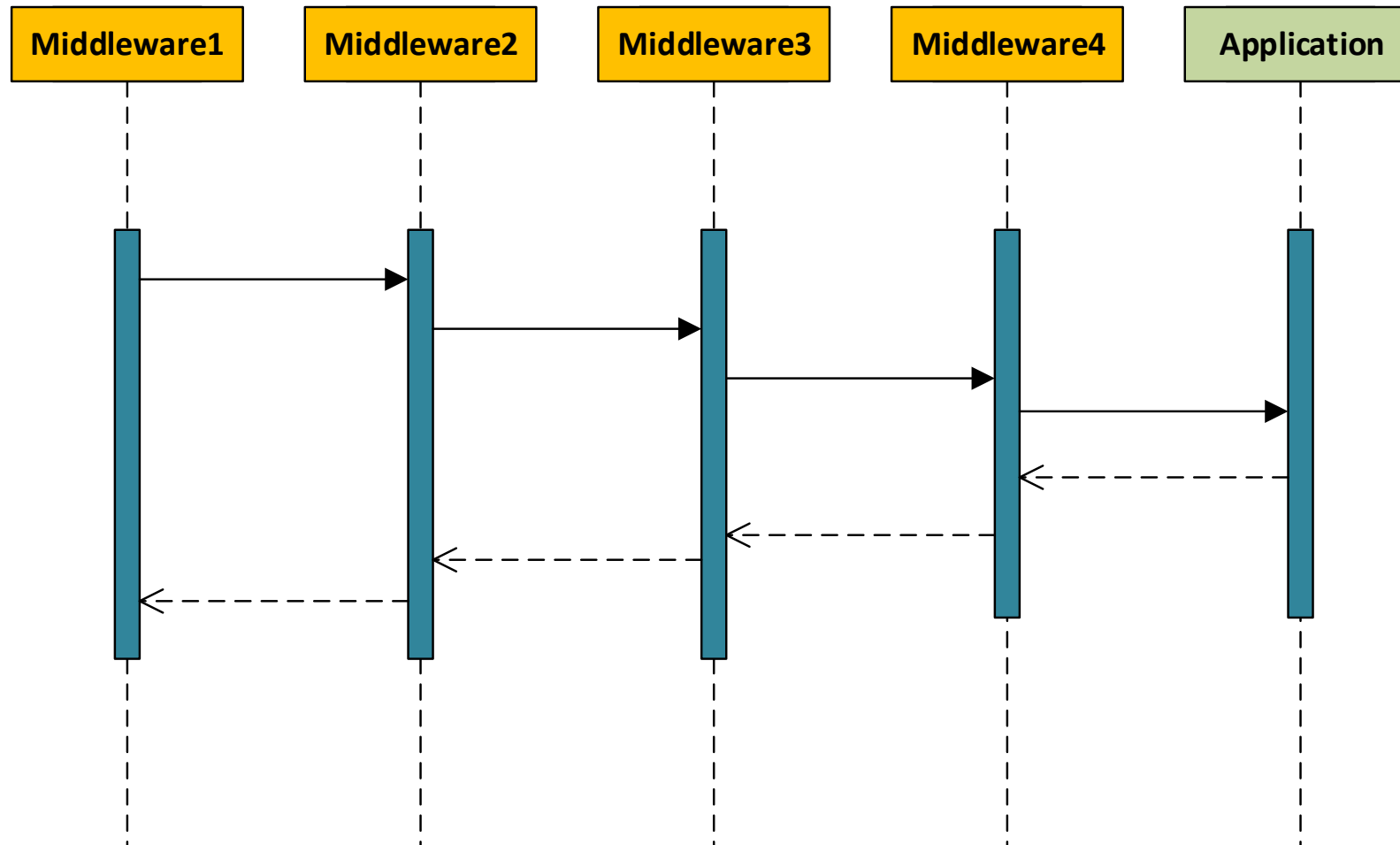
Современные платформы прикладной разработки

MIDDLEWARE

Middleware

Middleware - это программное обеспечение, которое собирается в конвейер приложения для обработки запросов и ответов.

Конвейер Middleware



Middleware

Каждый компонент:

- Выбирает, следует ли передавать запрос следующему компоненту в конвейере.
- Может выполнять работу до и после следующего компонента в конвейере.

Middleware

Цепочка нескольких компонентов создается с помощью метода `Use(context, next)`.

Параметр `"next"` представляет следующий делегат в конвейере. Вы можете закоротить конвейер, не вызывая следующий делегат.

Пример класса Middleware

```
public class MyMiddleware
{
    RequestDelegate _next;
    public MyMiddleware(RequestDelegate next)
    {
        _next = next;
    }
    public async Task Invoke(HttpContext context)
    {
        // Предварительные действия
        await _next.Invoke(context);
        // Завершающие действия
    }
}
```

Добавление middleware в конвейер

```
app.UseMiddleware<MyMiddleware>();
```

Создание расширяющего метода

```
public static class MiddlewareExtensions
{
    public static IApplicationBuilder UseMyMiddleware(this
IApplicationBuilder builder)
    {
        return builder.UseMiddleware<MyMiddleware>();
    }
}
```

Использование расширяющего метода

```
app.UseMyMiddleware();
```