

Alice

25.10.02 Init and Regular Function. How to place it in the Asp.Net Core

AspNetCore,RazorPage. У меня проект AspNetЦЩку 9.0 Пустая Страница Index.cshtml.cs выглядит так: using Microsoft.AspNetCore.Mvc.RazorPages; namespace RazorPageClearProj.Pages { public class IndexModel : PageModel { private readonly ILogger<IndexModel> _logger; public IndexModel(ILogger<IndexModel> logger) { _logger = logger; } public void OnGet() { } } } У меня в Проекте есть две функции в зарегистрированном SingleTone классе. В этом классе у меня две функции Init() - ее необходимо запустить только один раз в самом первом прохождении по коду выше. RegularWork. - эту функцию необходимо запускать в каждый раз, когда этот код работает. Код public class IndexModel : PageModel { private readonly ILogger<IndexModel> _logger; public IndexModel(ILogger<IndexModel> logger) { _logger = logger; } public void OnGet() { } } В каких местах.частях кода мне разместить мои функции ? - для однократного запуска - для регулярного запуска ?

Для решения этой задачи в ASP.NET Core с Razor Pages есть несколько подходов, как разместить ваши функции Init() и RegularWork().

1. Однократный запуск (Init())

Есть несколько вариантов, где можно разместить код однократного запуска:

- В методе ConfigureServices в Startup.cs (или Program.cs в новых проектах):

csharp

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddSingleton<MySingletonClass>(provider =>
    {
        var instance = new MySingletonClass();
        instance.Init(); // вызываем Init() при создании экземпляра
        return instance;
    });
}
```

- В конструкторе Singleton-класса — если Init() должен инициализировать внутренние параметры самого класса.
- В методе Configure (Startup.cs), если инициализация связана с настройкой middleware или HTTP-конвейера.
- В статическом конструкторе Singleton-класса, если это допустимо по логике приложения.

2. Регулярный запуск (RegularWork())

Для регулярного вызова на каждой странице можно использовать следующие подходы:

- В методе OnGet() (или OnPost(), OnPut() и т.д.) вашей PageModel:

csharp

```
public class IndexModel : PageModel
{
    private readonly MySingletonClass _singleton;

    public IndexModel(MySingletonClass singleton)
    {
        _singleton = singleton;
    }

    public void OnGet()
    {
        _singleton.RegularWork(); // вызывается при каждом запросе
    }
}
```

- Использовать middleware для автоматического вызова перед обработкой каждого запроса.
- Создать базовый класс для PageModel, где будет вызываться RegularWork(), и наследовать все страницы от него:

csharp

```
public abstract class BasePageModel : PageModel
{
    protected MySingletonClass _singleton;

    public BasePageModel(MySingletonClass singleton)
    {
        _singleton = singleton;
    }

    public override async Task OnInitializeAsync()
    {
        await base.OnInitializeAsync();
        _singleton.RegularWork();
    }
}
```

- Использовать фильтры (ActionFilters или PageFilters) для автоматического вызова функции перед/после выполнения действий.

Дополнительные рекомендации:

- Убедитесь, что Singleton-класс правильно зарегистрирован в DI-контейнере (через `AddSingleton`).

- Если `RegularWork()` выполняет длительные операции — рассмотрите асинхронный вариант (`async/await`).
- Для сложных сценариев инициализации можно использовать паттерн `Lazy Initialization` или `Factory Pattern`.
- Следите за потокобезопасностью при работе с `Singleton`-классом, особенно если `RegularWork()` модифицирует общие данные.

При выборе подхода ориентируйтесь на архитектуру вашего проекта и требования к производительности.

Мне метод `Init()` - необходимо запустить тоже на странице `RazorPage`, потому что этот метод тоже работает со старницей-HTML - он отрисовывает Первичную картинку.

В таком случае есть несколько подходов, как запустить метод `Init()` на `Razor Page`, который отвечает за отрисовку первичной картинки:

1. Запуск в конструкторе `PageModel`

Можно вызвать `Init()` в конструкторе `IndexModel`, если инициализация должна произойти при создании модели страницы:

```
csharp
public class IndexModel : PageModel
{
    private readonly MySingletonClass _singleton;

    public IndexModel(MySingletonClass singleton)
    {
        _singleton = singleton;
        _singleton.Init(); // вызов при создании модели
    }

    public void OnGet()
    {
        _singleton.RegularWork();
    }
}
```

2. Запуск в методе `OnGet()` или `OnPost()`

Если инициализация должна происходить при первом запросе к странице, можно использовать метод обработки запроса:

```
csharp
public void OnGet()
{
```

```

if (!_isInitialized)
{
    _singleton.Init();
    _isInitialized = true;
}
_singleton.RegularWork();
}

```

Здесь `_isInitialized` — это флаг, который гарантирует однократный запуск `Init()`.

3. Использование флага в Singleton-классе

Добавьте в `MySingletonClass` внутренний флаг инициализации:

```

csharp
public class MySingletonClass
{
    private bool _isInitialized;

    public void Init()
    {
        if (!_isInitialized)
        {
            // логика отрисовки первичной картинки
            _isInitialized = true;
        }
    }
}

```

4. Использование `OnInitializeAsync()`

Можно переопределить метод `OnInitializeAsync()`, который вызывается до всех других методов обработки:

```

csharp
public override async Task OnInitializeAsync()
{
    await base.OnInitializeAsync();
    _singleton.Init();
    _singleton.RegularWork();
}

```

5. Создание базового класса для всех страниц

Если `Init()` нужен на нескольких страницах, создайте базовый класс:

```

csharp
public abstract class BasePageModel : PageModel
{
    protected MySingletonClass _singleton;
}

```

```

public BasePageModel(MySingletonClass singleton)
{
    _singleton = singleton;
    _singleton.Init(); // однократная инициализация
}
}

```

6. Использование ViewComponent

Если отрисовка картинки — это часть представления, можно вынести логику в **ViewComponent**:

- создать компонент для отрисовки;
- зарегистрировать его в DI;
- вызвать в разметке **.cshtml** через **@await**
Component.InvokeAsync("MyComponent").

7. Middleware для инициализации

Если инициализация связана с глобальным состоянием приложения, можно создать middleware, который запустит **Init()** при первом запросе.

Важные моменты:

- убедитесь, что **MySingletonClass** зарегистрирован как синглтон в **Startup.cs** или **Program.cs**;
- проверьте потокобезопасность, если **Init()** модифицирует общие ресурсы;
- если отрисовка картинки требует асинхронных операций — используйте **async/await**;
- следите за производительностью: инициализация не должна сильно замедлять загрузку страницы.

Дополнительно можно использовать паттерн Lazy Initialization, чтобы отложить инициализацию до первого обращения к ресурсу.

В каком месте кода определяется и инициализируется переменная **_isInitialized** 2. Запуск в методе **OnGet()** или **OnPost()** Если инициализация должна происходить при первом запросе к странице, можно использовать метод обработки запроса: **csharp public void OnGet() { if (!_isInitialized) { _singleton.Init(); _isInitialized = true; } _singleton.RegularWork(); } ?**

Переменная **_isInitialized** в данном контексте определяется и инициализируется в классе **IndexModel** (или другом классе **PageModel**) проекта **Razor Pages**. Обычно это делается в одном из следующих мест:

1. В качестве поля класса с начальной инициализацией:

```
csharp
public class IndexModel : PageModel
{
    private readonly MySingletonClass _singleton;
    private bool _isInitialized = false; // начальная инициализация

    public IndexModel(MySingletonClass singleton)
    {
        _singleton = singleton;
    }

    public void OnGet()
    {
        if (!_isInitialized)
        {
            _singleton.Init();
            _isInitialized = true;
        }
        _singleton.RegularWork();
    }
}
```

2. В конструкторе класса, если нужно задать начальное состояние при создании объекта:

```
csharp
public IndexModel(MySingletonClass singleton)
{
    _singleton = singleton;
    _isInitialized = false; // установка начального значения
}
```

3. В блоке объявления полей класса, если требуется более сложная логика инициализации.

Объяснение работы кода в методе `OnGet()`:

Когда пользователь делает HTTP GET-запрос к странице, вызывается метод `OnGet()`. Внутри этого метода происходит проверка:

1. Если `_isInitialized` равен `false` (инициализация ещё не была выполнена):
 - вызывается метод `Init()` у синглтона;
 - флаг `_isInitialized` устанавливается в `true`, сигнализируя о том, что инициализация уже выполнена.

2. После этого (независимо от значения `_isInitialized`) вызывается метод `RegularWork()`, который будет выполняться при каждом запросе к странице.

Важные нюансы:

- Флаг `_isInitialized` гарантирует, что метод `Init()` будет вызван только один раз — при первом обращении к странице.
- Метод `OnGet()` — это обработчик HTTP GET-запросов в Razor Pages. Его можно заменить на `OnPost()`, если инициализация должна происходить при POST-запросе.
- Асинхронная версия: если `Init()` содержит асинхронные операции, можно использовать асинхронный обработчик: `OnGetAsync()` или `OnPostAsync()`.
- Потокобезопасность: если приложение многопоточное, стоит предусмотреть синхронизацию доступа к `_isInitialized`.
- Альтернативные подходы: вместо флага можно использовать другие способы отслеживания состояния (например, проверку наличия определённых данных в хранилище или базе данных).

Также стоит помнить, что HTTP является stateless-протоколом, поэтому если нужно сохранять состояние между запросами, может потребоваться использовать сессии, куки или другие механизмы хранения данных.

```
namespace RazorPageClearProj.Pages { public class IndexModel : PageModel { private
readonly ILogger<IndexModel> _logger; private bool _isInitialized = false; // начальная
инициализация public IndexModel(ILogger<IndexModel> logger, bool isInitialized) {
_logger = logger; _isInitialized = false; } public void OnGet() { // if(_isInitialized) == false;
} } } ----- InvalidOperationException: Unable to resolve service for type
'System.Boolean' while attempting to activate
'RazorPageClearProj.Pages.IndexModel'.
```

Данная ошибка `InvalidOperationException: Unable to resolve service for type`

`'System.Boolean' while attempting to activate`

`'RazorPageClearProj.Pages.IndexModel'` возникает потому, что система внедрения зависимостей (Dependency Injection) в ASP.NET Core не может найти сервис типа `Boolean` для передачи в конструктор `IndexModel`.

Причина проблемы в том, что вы пытаетесь передать `bool isInitialized` в конструктор, хотя такой сервис не зарегистрирован в контейнере зависимостей.

Вот как можно исправить ситуацию:

1. Убрать параметр `bool isInitialized` из конструктора.

Флаг `_isInitialized` должен быть внутренней переменной класса, а не приниматься извне.

Исправленный код:

```
csharp
namespace RazorPageClearProj.Pages
{
    public class IndexModel : PageModel
    {
        private readonly ILogger<IndexModel> _logger;
        private bool _isInitialized = false; // начальная инициализация

        public IndexModel(ILogger<IndexModel> logger)
        {
            _logger = logger;
        }

        public void OnGet()
        {
            if (!_isInitialized)
            {
                // код инициализации
                _isInitialized = true;
            }
        }
    }
}
```

2. Использовать внутреннюю логику для управления состоянием.

Флаг `_isInitialized` должен управляться внутри класса, а не передаваться извне.

3. Альтернативные способы управления состоянием:

- использовать статическую переменную (с учётом потокобезопасности);
- хранить состояние в базе данных или кэше;
- использовать сессии или куки (если состояние нужно сохранять между запросами).

4. Если нужно передавать какие-то параметры, их следует:

- зарегистрировать в контейнере зависимостей;
- использовать параметры запроса (Query String);

- передавать через модель или ViewModel.

5. Проверить регистрацию сервисов в файле `Startup.cs` или `Program.cs`.

Убедитесь, что все необходимые сервисы зарегистрированы

через `services.AddSingleton()`, `services.AddScoped()` или `services.AddTransient()`.

6. Использовать асинхронные методы, если инициализация требует асинхронных операций:

```
csharp
public async Task OnGetAsync()
{
    if (!_isInitialized)
    {
        await InitializeAsync();
        _isInitialized = true;
    }
}
```

7. Рассмотреть паттерн Singleton или другие шаблоны проектирования, если инициализация должна быть глобальной для всего приложения.

8. Добавить логирование для отладки процесса инициализации:

```
csharp
_logger.LogInformation("Инициализация выполнена");
```

Таким образом, флаг `_isInitialized` должен быть внутренним состоянием класса `IndexModel`, а не внешним параметром конструктора.