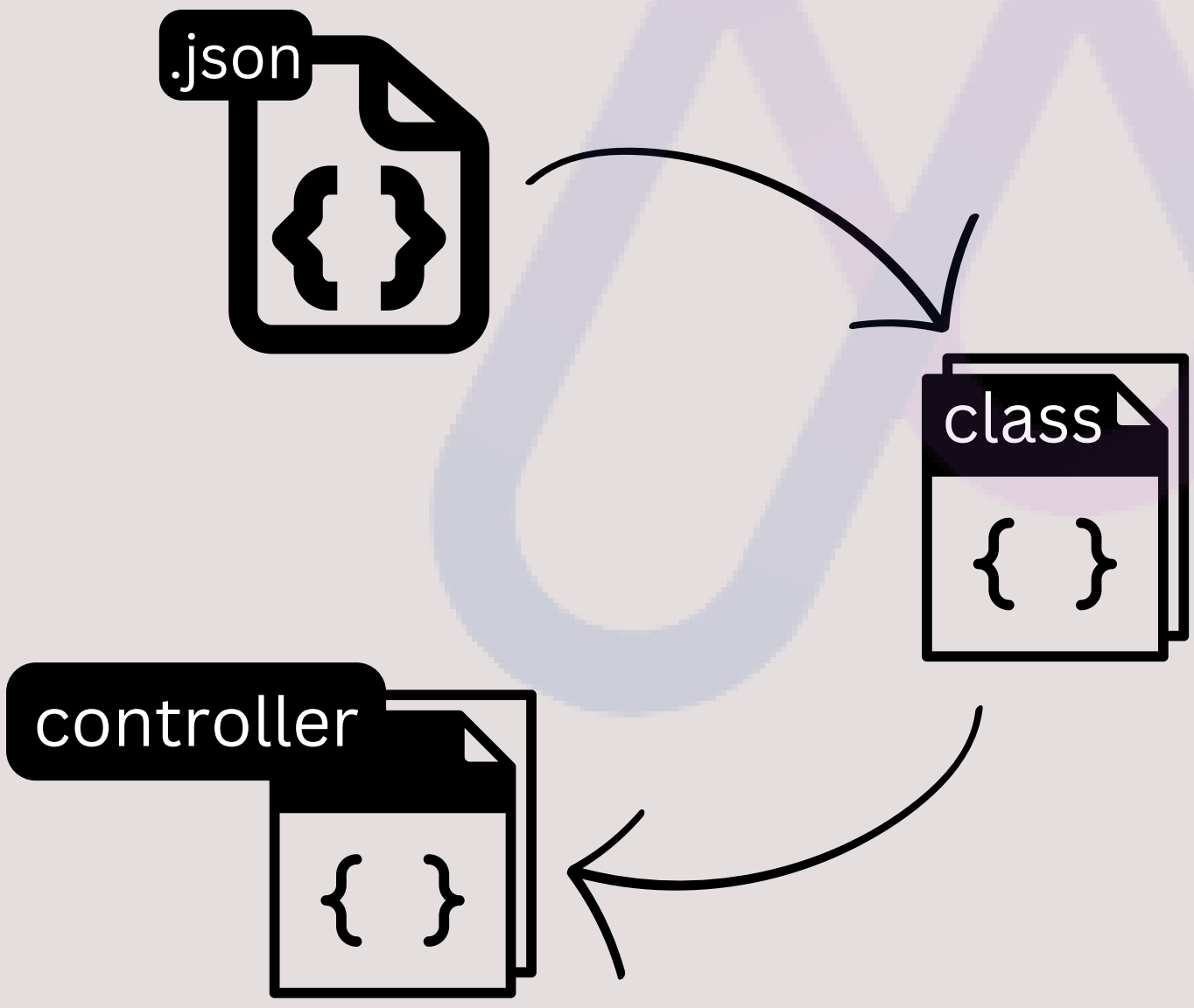
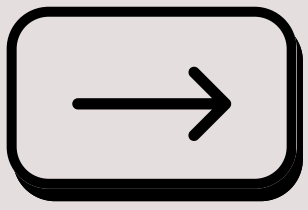
Ajay Patel

.NET

Easier Than You THINK!

Options Pattern





Ajay Patel

Create a class that represents the settings you want to configure. For example, if you have settings related to Smtp information.

```
public class SmtpOptions{
    public string? Host { get; set; }
    public int Port { get; set; }
    public string? Username { get; set; }
    public string? Password { get; set; }
}
```

Ajay Patel

3

Add your settings in the appsettings.json file under a specific section.

```
"SmtpOptions": {
    "Host": "smtp.example.com",
    "Port": 587,
    "Username": "user@example.com",
    "Password": "securepassword"
}
```

In the Program.cs, bind the configuration section to your class.

```
builder.Services.Configure<SmtpOptions>(builder.Configuration.GetSection(nameof(SmtpOptions)));
```

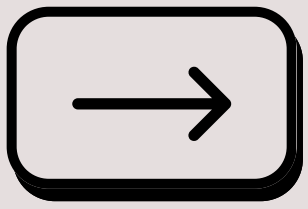
Ajay Patel

4

In your controller or service, inject IOptions<SmtpOptions> to access the settings

```
public class EmailService
{
    private readonly SmtpOptions _smtpOptions;
    public EmailService(IOptions<SmtpOptions> smtpOptions)
    {
        _smtpOptions = smtpOptions.Value;
    }
    //Use _smtpOptions to send emails
    public void SendEmail(){
        SmtpClient smtpClient = new SmtpClient{
            Host = _smtpOptions.Host,
            Port = _smtpOptions.Port
            ...
        }
    }

}
```



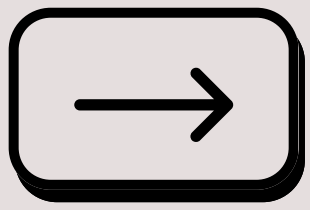
Ajay Patel

Using IOptionsSnapshot:

If you want to change the PORT and Password, we need to restart the application for changes to take effect, because **IOptions<T> doesn’t support live reloading.**

However, a more efficient option exists, known as **IOptionsSnapshot**.

IOptionsSnapshot registered as a **scoped service** and it contains the values just for the lifetime of a request. we should use it only with scoped and transient dependencies. We cannot inject it into singleton services!



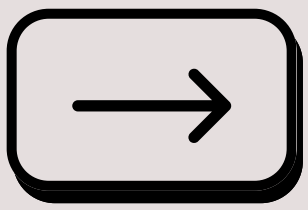
Ajay Patel

6

Use **IOptionsSnapshot** to Read the Updated Configuration

```
public class EmailService
{
    private readonly SmtpOptions _smtpOptions;
    public EmailService(IOptionsSnapshot<SmtpOptions> smtpOptions)
    {
        _smtpOptions = smtpOptions.Value;
    }
    //Use _smtpOptions to send emails
    public void SendEmail(){
        SmtpClient smtpClient = new SmtpClient{
            Host = _smtpOptions.Host,
            Port = _smtpOptions.Port
            ...
        }
    }

}
```



Ajay Patel

Using IOptionsMonitor:

IOptionsMonitor is registered as a **singleton service.**

So, if your service is registered as a singleton than you should use IOptionsMonitor.

In addition, If you want configuration values to be available instantly than you can use IOptionsMonitor.

IOptionsSnapshot is not suitable to be injected into services registered as a singleton, it will throw an error.

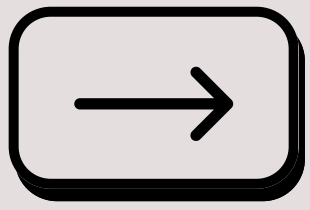
Ajay Patel

8

Using **IOptionsMonitor** for Singleton Services:

```
public class EmailService
{
    private readonly IOptionsMonitor<SmtpOptions> _smtpOptions;
    public EmailService(IOptionsMonitor<SmtpOptions> smtpOptions)
    {
        _smtpOptions = smtpOptions;
    }
    //Use _smtpOptions to send emails
    public void SendEmail(){
        SmtpClient smtpClient = new SmtpClient{
            Host = _smtpOptions.CurrentValue.Host,
            Port = _smtpOptions.CurrentValue.Port
            ...
        };
    }
}
```

IOptionsMonitor uses the **CurrentValue** instead of Value retrieving the configuration values.



Ajay Patel

9

Named Options:

Named options aren’t a feature we frequently use, but there’s a particular scenario where they can be quite useful. For instance, consider a configuration like this one:

```
"SmtpOptionsForAdmin": {
    "Host": "smtp.admin.com",
    "Port": 587,
    "Username": "admin@example.com",
    "Password": "adminpassword"
},
"SmtpOptionsForUser": {
    "Host": "smtp.user.com",
    "Port": 587,
    "Username": "user@example.com",
    "Password": "userpassword"
},
```

Ajay Patel

We use the same configuration structure for the various sections within the appsettings.json.

Both the ‘SmtpOptionsForAdmin’ and the ‘SmtpOptionsForUser’ share **identical configuration properties.**

Rather than creating another configuration class identical to our existing SmtpOptions, we can reuse that class to map different sections, simply naming them differently to distinguish between them.

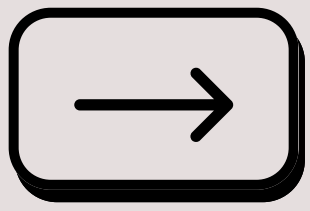
Ajay Patel

In our **Program.cs** class we should configure:

```
builder.Services.Configure<SmtpOptions>("AdminSmtp",builder.Configuration.GetSection("SmtpOptionsForAdmin"));
builder.Services.Configure<SmtpOptions>("UserSmtp",builder.Configuration.GetSection("SmtpOptionsForUser"));
```

Now both sections are mapped to the same configuration class. **To call the specific option, we now use the Get() method.**

```
public class EmailService
{
    private readonly SmtpOptions _smtpOptions;
    public EmailService(IOptionsSnapshot<SmtpOptions> smtpOptions)
    {
        _smtpOptions = smtpOptions.Get("AdminSmtp");
    }
    //Use _smtpOptions to send emails
    public void SendEmail(){
        SmtpClient smtpClient = new SmtpClient{
            Host = _smtpOptions.Host,
            Port = _smtpOptions.Port
            ...
        };
    }
```



Ajay Patel

IOptions Vs. IOptionsSnapshot Vs. IOptionsMonitor

IOptions<T>:

- Is registered as a singleton service.
- Binds the configuration values only once at the registration, and returns the same values every time.
- Does not support named options

IOptionsSnapshot<T>:

- Registered as a scoped service
- Supports configuration reloading
- Values reload per request
- Supports named options

IOptionsMonitor<T>:

- Registered as a singleton service
- Supports configuration reloading
- Values are cached and reloaded immediately
- Supports named options

Ajay Patel

.NET

Knowledge is contagious, let's spread it!

DO YOU LIKE THIS POST?

REPOST IT!

THANKS FOR READING