Applying the Options Pattern

Steve Gordon

.NET Engineer and Microsoft MVP

@stevejgordon | www.stevejgordon.co.uk

Overview

Introduce the options pattern

- IOptions<T>
- IOptionsSnapshot<T>
- IOptionsMonitor<T>

Use named options

Apply options validation

Unit testing dependent types

Apply the options pattern

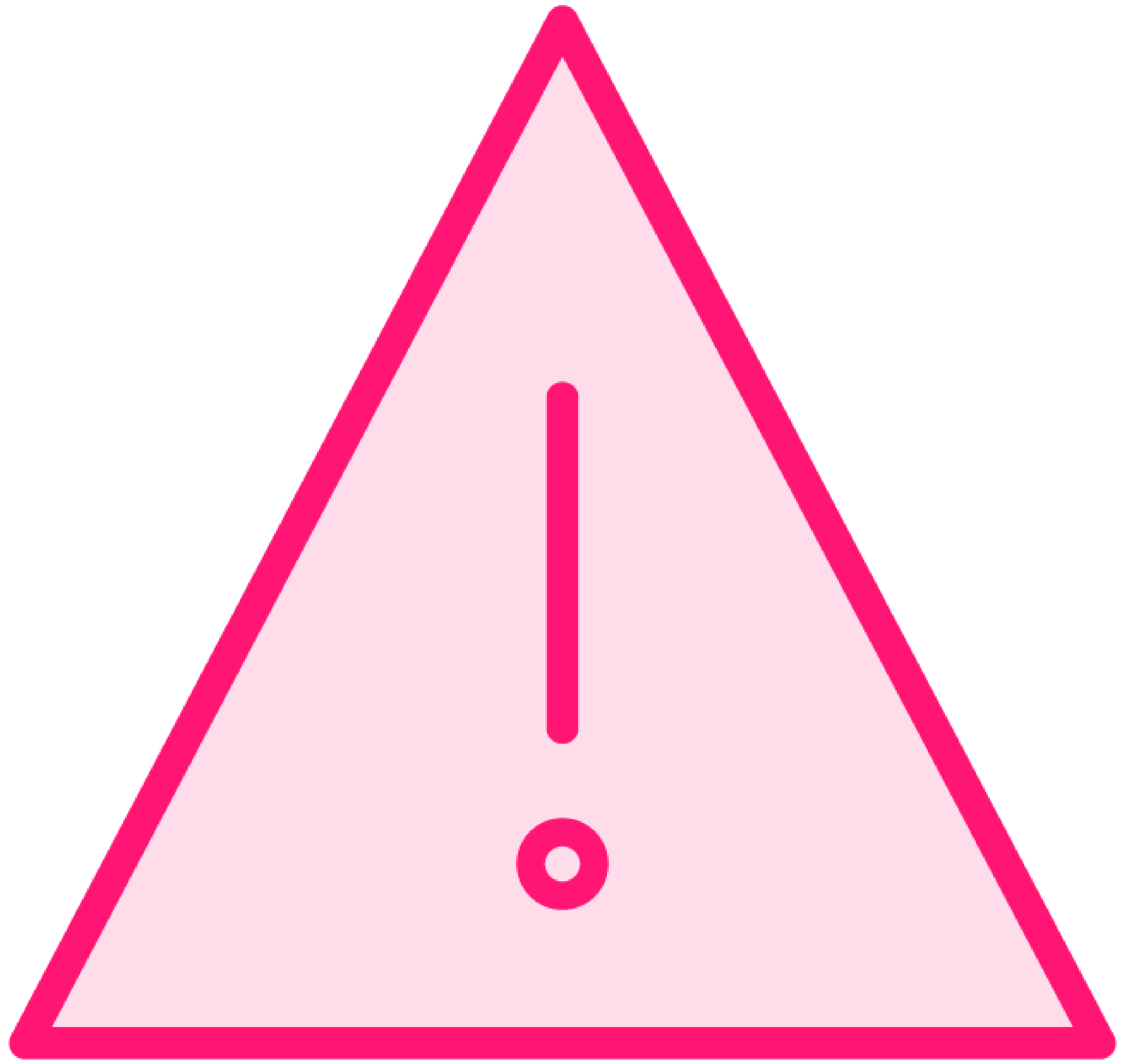
Bind configuration to strongly typed options classes

Inject options with IOptions<T>

Benefits

Define classes representing related properties

Apply the single responsibility principle



Microsoft.Extensions.Options

Reloading with IOptionsSnapshot<T>

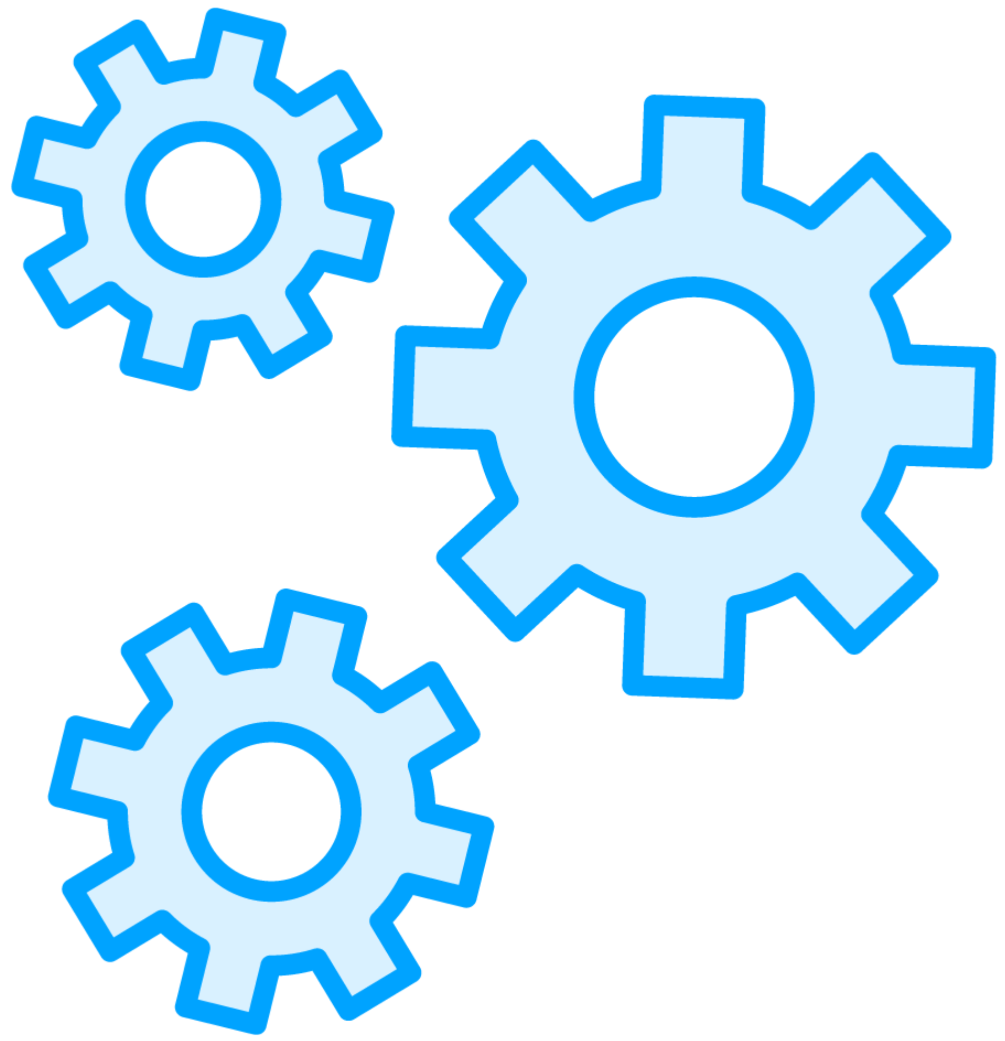
appSettings.json

Loaded by the JSON configuration provider

Supports change notifications when the file is modified

Accessing options via IOptionsMonitor<T>

IOptions



Implementations are registered with D.I. using the singleton lifetime

A single instance is created and reused

The same resolved instance is injected to all dependent services

Changes to the configuration are not reflected until the application restarts

Take care when allowing configuration to be reloaded whilst an application is running

Non-critical feature flags may be fairly safe to reload

Risk may be introduced when reloading configuration values, such as database connection strings

Consider carefully if reloading is safe for the consumed options class

Learn More

Dependency Injection in ASP.NET Core 6

Steve Gordon

Using named options

- Defining named options
- Consuming named options

Choosing Between the Options Interfaces

IOptions<T>

Does not support options reloading

Registered as a singleton in D.I. container

Values bound when first used

Can be injected into all service lifetimes

Does not support named options

IOptionsSnapshot <T>

Supports reloading of configuration

Registered as scoped in D.I. container

Values may reload per request

Can not be injected into singleton services

Supports named options

IOptionsMonitor<T>

Supports reloading of configuration

Registered as a singleton in D.I. container

Values are reloaded immediately

Can be injected into all service lifetimes

Supports named options

Choosing an Options Interface

	Use in singletons	Supports reloading	Named options
IOptions	✓	✗	✗
IOptionsSnapshot	✗	✓	✓
IOptionsMonitor	✓	✓	✓

Options validation

- Data annotation attributes

When is validation applied?

Ensuring options are valid at start up

Advanced validation techniques

- Define conditional validation logic
- Implement IValidateOptions

Learn More

Dependency Injection in ASP.NET Core 6

Steve Gordon

Validating named options

Forwarding to options via an interface

Unit testing types dependent on options classes

- Using Options.Create
- Mocking with Moq
- Using IServiceProvider

Summary

Options pattern

- Bound configuration to strongly-typed options

Options consumption

- IOptions<T>
- IOptionsSnapshot<T>
- IOptionsMonitor<T>

Used named options

Applied options validation

Unit testing

Up Next:

Working with Configuration Providers