

Timeout strategy CHEAT SHEET



Basics

This proactive resilience strategy **cancels the execution** if it **does not complete** within the specified timeout period.

You can configure the behaviour of the strategy via the **TimeoutStrategyOptions** object.

If the execution is canceled, then the strategy will throw **TimeoutRejectedException**.

Specify constant timeout

```
new ResiliencePipelineBuilder()
    .AddTimeout(new TimeoutStrategyOptions
    {
        Timeout = TimeSpan.FromSeconds(10)
    })
```

Specify constant timeout – short form

```
new ResiliencePipelineBuilder()
    .AddTimeout(TimeSpan.FromSeconds(10))
```

Specify dynamic timeout based on a context property

```
var waitKey = new ResiliencePropertyKey<bool>("shouldWaitLonger");

new ResiliencePipelineBuilder()
    .AddTimeout(new TimeoutStrategyOptions
    {
        TimeoutGenerator = args =>
        {
            var waitLonger = args.Context.Properties.GetValue(waitKey, false);
            return ValueTask.FromResult(TimeSpan.FromSeconds(waitLonger ? 20 : 10));
        }
    })
```

Specify dynamic timeout asynchronously

```
new ResiliencePipelineBuilder()
    .AddTimeout(new TimeoutStrategyOptions
    {
        TimeoutGenerator = async args => await GetTimeoutLimitAsync()
    })
```

Specify delegate for timeout notification

```
new ResiliencePipelineBuilder()
    .AddTimeout(new TimeoutStrategyOptions
    {
        OnTimeout = static args =>
        {
            Console.WriteLine($"Method cancelled after {args.Timeout.TotalSeconds} sec.");
            return default;
        }
    })
```

Specify asynchronously delegate for timeout notification

```
new ResiliencePipelineBuilder()
    .AddTimeout(new TimeoutStrategyOptions
    {
        OnTimeout = async args => await NotifyAsync(args.Timeout)
    })
```