Polly: Eine "Resilience" Bibliothek

(Kategorie: nützliche NuGet Pakete)

Patrick Drechsler
Redheads Ltd.

2020-06-22

Resilience deutsch

All News Images Videos Shopping More Settings Tools

About 34.700.000 results (0,31 seconds)

English – detected German

Resilience

Elastizität

Translations of **resilience**

noun

die Elastizität
elasticity, resilience, resiliency, flexibility, suppleness, springiness

die Spannkraft
resilience, tone, vigor, vigour, tonus, tension

die Unverwüstlichkeit
resilience, resiliency, robustness

RESILIENCE ALS "1ST CLASS CITIZEN"

Fehlerhaftes Verhalten von anderen Diensten

- als "normal" betrachten und
- flexibel darauf reagieren

*Polly is a .NET **resilience** and transient-fault-handling library that **allows developers to express policies such as Retry, Circuit Breaker, Timeout**, Bulkhead Isolation, and Fallback **in a fluent and thread-safe manner**.*

POLLY PROJECT

- stabiles, gepflegtes Repo
- große Community
- 7k Sterne auf Github
- https://github.com/App-vNext/Polly
- http://www.thepollyproject.org

WAS SIND POLICIES?

Policies beschreiben das Verhalten, wenn was schief geht

BEISPIEL "POLICIES"

- **Timeout**
  - "Antwort dauert länger als 10sec"
- **Retry**
  - "keine Antwort -> nochmal probieren"
- **Circuit Breaker**
  - "verhindere weiter Anfragen, damit sich der Dienst erholen kann"
  - "wenn sich Dienst wieder erholt hat -> weitermachen"
- ...

POLLY STANDARDVORGEHEN

- Policy definieren
- eigentliche Funktion in Policy "einpacken"

BEISPIEL USE-CASE

```
public class BusinessLogic
{
  private readonly IFlakyService _srv;

  public BusinessLogic(IFlakyService service) ⇒ _srv = service;

  public int CallFlakyMethod()
  {
    return _srv.SlowMethod(); // ← SLOW!!
  }
}
```

BEISPIEL 1: RETRY

RETRY-POLICY DEFINIEREN

```
var _policy = Policy
    .Handle<Exception>()
    .WaitAndRetry(3, x ⇒ TimeSpan.FromSeconds(2));
```

- `Handle`: welche Exception?
- `WaitAndRetry`: Policy "Strategie"

POLICY ANWENDEN

```
public int CallFlakyMethod()
{

  return _policy.Execute(() ⇒ _srv.SlowMethod());
}
```

BEISPIEL 2: CIRCUIT BREAKER

```
var circuitBreakerPolicy = Policy
    .Handle<Exception>()
    .CircuitBreaker(1, TimeSpan.FromSeconds(1),
        (ex, t) ⇒
        {
            Log.Information("Circuit broken!");
        },
        () ⇒
        {
            Log.Information("Circuit Reset!");
        });
```

```
_circuitBreakerPolicy = Policy
    .Handle<Exception>() // PolicyBuilder
    .CircuitBreaker( exceptionsAllowedBeforeBreaking: 1,  durationOfBreak: TimeSpan.FromSeconds(1),
        onBreak: (ex :Exception , t :TimeSpan ) ⇒
        {
            Log.Information( messageTemplate: "Circuit broken!")
        },
        onReset: () ⇒
        {
            Log.Information( messageTemplate: "Circuit Reset!");
        }); // CircuitBreakerPolicy
```

LIVE-DEMO

Policies können

- sehr feingranular definiert werden
- miteinander kombiniert werden

FAZIT

Polly einsetzen, wenn man oft und/oder komplexe Policies einsetzt.