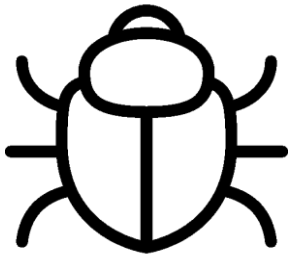
SINGULARIS LAB

Polly. Make it robust

Анатолий Крыжановский

503. Server unavailable



```
var retry = 3;
ClientData _data;
try
{
while(retry > 0)
{                   data = _client.Execute<ClientData>(request);
}                   try
catch(Exception e)  {
{                                   data = _client.Execute<ClientData>(request);
                                    break;
                    _logger.Error(e, "Something goes wrong");
}                   }
                    catch(Exception e)
                    {
                                    _logger.Error(e, "Something goes wrong");
                                    retry--;
                    }
}
```

Что? Где? Зачем?

- Библиотека для повышения отказоустойчивости вашего приложения при обработки временных сбоев при помощи политик.
- Реализуемые политики:
  - Retry
  - Circuit Breaker
  - Timeout
  - Bulkhead
  - Fallback
  - NoOp
  - Cache

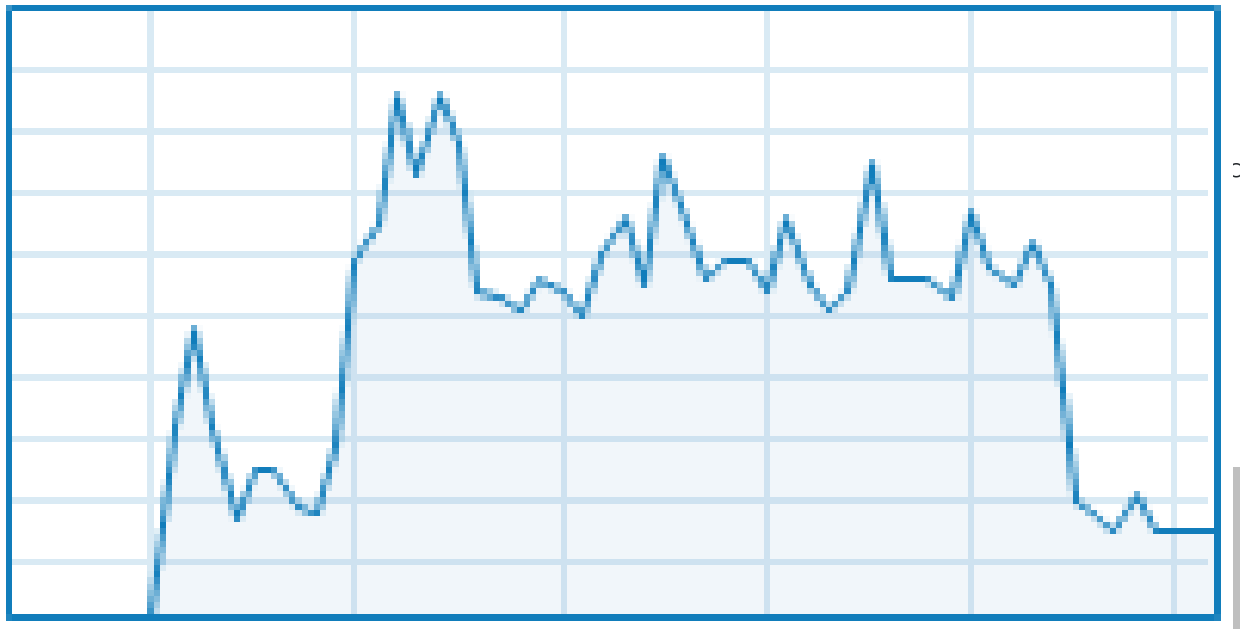
Retry

```
var policy = Policy
    .Handle<Exception>()
    .Retry(
        5,
        (exception, attempt) =>
            Logger.Log($"Attempt {attempt} failed with message: {exception.Message}", ConsoleColor.Red);
        );
var result = policy.Execute(() => client.SimulateIoException(5), cnt);
Logger.Log($"Successful executed on {result} attempt", ConsoleColor.Green);
```

```
[11:07:17] Attempt 1 failed with message: Failed on 1 attempt
[11:07:17] Attempt 2 failed with message: Failed on 2 attempt
[11:07:17] Attempt 3 failed with message: Failed on 3 attempt
[11:07:17] Attempt 4 failed with message: Failed on 4 attempt
[11:07:17] Successful executed on 5 attempt
```

RetryForever

```
var policy = Policy
    .Handle<Exception>()
    .RetryForever(
        (Exception exception,
            Logger.Log($"Attem                                    olor.Red);
        );
var result = policy.Execute(()
Logger.Log($"Successful execut
```



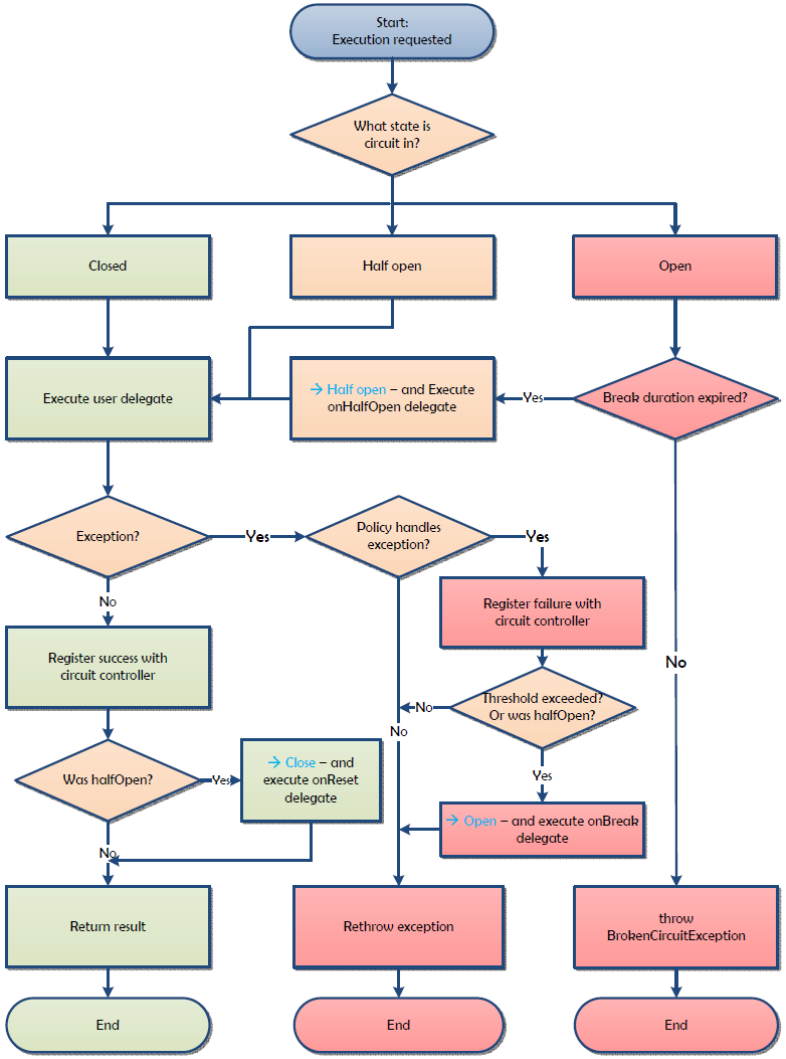
```
[11:07:14] Attempt 1 failed wi
[11:07:14] Attempt 2 failed wi
…
[11:07:16] Attempt 121 failed with message: Failed on 121 attempt
```

WaitAndRetry

```
var jitter = new Random();
var policy = Policy
    .Handle<Exception>()
    .WaitAndRetry(
        5,
       (attempt, exception, context) => TimeSpan.FromSeconds(Math.Pow(2, attempt) + jitter.NextDouble()),
       (exception, timeSpan, attempt, context) =>
           Logger.Log($"Attempt {attempt} failed with message: {exception.Message}", ConsoleColor.Red);
       );
var result = policy.Execute(() => client.SimulateIoException(3));
Logger.Log($"Successful executed on {result} attempt", ConsoleColor.Green);
```

```
[11:07:32] Attempt 1 failed with message: Failed on 1 attempt
[11:07:34] Attempt 2 failed with message: Failed on 2 attempt
[11:07:39] Successful executed on 3 attempt
```

CircuitBreaker



CircuitBreaker

```
var breaker = Policy
    .Handle<Exception>()
    .CircuitBreaker(
         3,
         TimeSpan.FromSeconds(2),
        (exception, interval) => Logger.Log($"Break for {interval}. Reason: {exception.Message}", ConsoleColor.Yellow),
        () => Logger.Log("CircuitBreaker closed", ConsoleColor.Yellow),
        () => Logger.Log("CircuitBreaker half-opened", ConsoleColor.Yellow));
var retry = Policy
    .Handle<Exception>(e => !(e is BrokenCircuitException))
    .WaitAndRetryForever(attempt => TimeSpan.FromMilliseconds(1000));
var policy = Policy.Wrap(retry, breaker);
var success = false;
while (!success)
{
    try
    {
        var result = policy.Execute(() => client.SimulateIoException(5));
        Logger.Log($"Successful executed on {result} attempt", ConsoleColor.Green);
        success = true;
    }
    catch (Exception e)
    {
        Logger.Log("CircuitBreaker opened", ConsoleColor.Red);
    }
    Thread.Sleep(TimeSpan.FromSeconds(1));
}
```

CircuitBreaker

```
[13:07:24] Perform request (0)
[13:07:25] Perform request (1)
[13:07:26] Perform request (2)
[13:07:26] Break for 00:00:02. Reason: Failed on 3 attempt
[13:07:27] CircuitBreaker opened
[13:07:28] CircuitBreaker half-opened
[13:07:28] Perform request (3)
[13:07:28] Break for 00:00:02. Reason: Failed on 4 attempt
[13:07:29] CircuitBreaker opened
[13:07:30] CircuitBreaker half-opened
[13:07:30] Perform request (4)
[13:07:30] CircuitBreaker closed
[13:07:30] Successful executed on 5 attempt
```

AdvancedCircuitBreaker

- реакция на процент ошибок;
- скользящее окно;
- учет нагрузки.

Timeout

```
try
{
    var policy = Policy
        .Timeout(
            2,
            TimeoutStrategy.Pessimistic,
            (context, interval, task, e) => Logger.Log($"Timeout triggered after {interval}", ConsoleColor.Red));
    var result = policy.Execute(() => client.SimulateTimeout());
    Logger.Log($"Successful executed on {result} attempt", ConsoleColor.Green);
}
catch (Exception e)
{
}
Thread.Sleep(5000);
```

[13:07:09] Perform request (0)
[13:07:11] Timeout triggered after 00:00:02
[13:07:14] But task still run

Timeout

```
try
{
    var policy = Policy
        .TimeoutAsync(
            2,
            TimeoutStrategy.Optimistic,
            (context, interval, task, e) => {
                Logger.Log($"Timeout triggered after {interval}", ConsoleColor.Red);
                return Task.CompletedTask;
            });
    var result = await policy.ExecuteAsync(async ct => await client.SimulateTimeoutAsync(ct), CancellationToken.None);
    Logger.Log($"Successful executed on {result} attempt", ConsoleColor.Green);
}
catch (Exception e) { }
Thread.Sleep(5000);
```

[13:07:06] Perform request (0)
[13:07:08] Timeout triggered after 00:00:02

Bulkhead

```
var policy = Policy
    .Bulkhead( 8, 1, context => Logger.Log($"Action rejected ({context["id"]})", ConsoleColor.Red));
var tasks = Enumerable
    .Range(0, 15)
    .Select(x => {
        var cnt = new Context{{"id", x}};
        return Task.Factory.StartNew(() =>
          policy.Execute((context) => client.SimulateFailedOrSuccess(x, x % 2 == 0), cnt)));
    })
    .ToArray();

await Task.WhenAll(tasks);
```

Bulkhead

[14:07:45] Execute 0 operation
[14:07:45] Execute 1 operation
[14:07:45] Execute 3 operation
[14:07:45] Execute 2 operation
[14:07:45] Execute 4 operation
[14:07:46] Execute 5 operation
[14:07:47] Execute 6 operation
[14:07:48] Execute 7 operation
[14:07:50] 2 operation success
[14:07:50] 3 operation failed
[14:07:50] 1 operation failed
[14:07:50] 0 operation success
[14:07:50] Execute 8 operation
[14:07:50] Execute 9 operation
[14:07:50] Execute 10 operation
[14:07:50] Execute 11 operation
[14:07:50] Action rejected (13)
[14:07:50] 4 operation success
[14:07:51] Execute 12 operation
[14:07:51] 5 operation failed
[14:07:52] Execute 14 operation
[14:07:52] 6 operation success
[14:07:53] 7 operation failed
[14:07:55] 8 operation success
[14:07:55] 9 operation failed
[14:07:55] 10 operation success
[14:07:55] 11 operation failed
[14:07:56] 12 operation success
[14:07:57] 14 operation success

Fallback

```
var fallback = Policy<int>
    .Handle<Exception>()
    .Fallback(42);

var retry = Policy<int>
    .Handle<Exception>()
    .Retry(5);

var policy = Policy.Wrap(fallback, retry);
var magicNumber = policy.Execute(() => client.SimulateFailedOrSuccess(0, false));
Logger.Log($"Magic number is: {magicNumber}", ConsoleColor.Green);
```

```
[14:07:28] Execute 0 operation
[14:07:33] 0 operation failed
[14:07:34] Execute 0 operation
[14:07:39] 0 operation failed
[14:07:39] Execute 0 operation
[14:07:44] 0 operation failed
[14:07:44] Execute 0 operation
[14:07:49] 0 operation failed
[14:07:49] Execute 0 operation
[14:07:54] 0 operation failed
[14:07:54] Execute 0 operation
[14:07:59] 0 operation failed
[14:07:59] Magic number is: 42
```

NoOp

```
var policy = Policy
    .NoOp();
var result = policy.Execute(() => client.SimulateFailedOrSuccess(1, true));
Logger.Log($"Success with result {result}", ConsoleColor.Green);
```

```
[15:07:03] Execute 1 operation
[15:07:08] 1 operation success
[15:07:08] Success with result 1
```

Cache

```
var memoryCache = new MemoryCache(new MemoryCacheOptions());
var cacheProvider = new Polly.Caching.Memory.MemoryCacheProvider(memoryCache);
var cache = Policy
    .Cache(
        cacheProvider,
        new RelativeTtl(TimeSpan.FromSeconds(2)),
        context => $"items_{context["id"]}",
        (context, key) => Logger.Log($"Get item from cache: {key}", ConsoleColor.Yellow),
        (context, key) => Logger.Log($"Miss item with key: {key}", ConsoleColor.Yellow),
        (context, key) => Logger.Log($"Put item to cache: {key}", ConsoleColor.Yellow),
        (context, key, e) => { },
        (context, key, e) => { });
var cnt = new Context {{"id", Guid.NewGuid()}};
Logger.Log("First call", ConsoleColor.Gray);
var value = cache.Execute((c) => client.SimulateFailedOrSuccess(1, true), cnt);
Logger.Log($"Result: {value} ", ConsoleColor.Green);
Logger.Log("Second call", ConsoleColor.Gray);
value = cache.Execute((c) => client.SimulateFailedOrSuccess(1, true), cnt);
Logger.Log($"Result: {value} ", ConsoleColor.Green);
cnt = new Context {{"id", Guid.NewGuid()}};
Logger.Log("Third call", ConsoleColor.Gray);
value = cache.Execute((c) => client.SimulateFailedOrSuccess(1, true), cnt);
Logger.Log($"Result: {value} ", ConsoleColor.Green);
```

Cache

```
[15:07:31] First call
[15:07:31] Miss item with key: items_faa2ec24-4e40-4a11-aaf1-a7c2e2a9c183
[15:07:31] Execute 1 operation
[15:07:36] 1 operation success
[15:07:36] Put item to cache: items_faa2ec24-4e40-4a11-aaf1-a7c2e2a9c183
[15:07:36] Result: 1
[15:07:36] Second call
[15:07:36] Get item from cache: items_faa2ec24-4e40-4a11-aaf1-a7c2e2a9c183
[15:07:36] Result: 1
[15:07:36] Third call
[15:07:36] Miss item with key: items_27bfb1e0-d7e9-48f6-8fe8-2b58d2ee6f0c
[15:07:36] Execute 1 operation
[15:07:41] 1 operation success
[15:07:41] Put item to cache: items_27bfb1e0-d7e9-48f6-8fe8-2b58d2ee6f0c
[15:07:41] Result: 1
```

Polly

- Async/sync
- Context
- PolicyRegistry

- https://github.com/App-vNext/Polly/wiki/

Вопросы?