Санкт-Петербургский государственный университет

Кафедра системного программирования

Rx.NET

Алина Летягина

st128889@spbu.ru

Санкт-Петербург
2025

Реактивное программирование: зачем?

- Традиционные подходы к асинхронности сложно поддерживать
  - Callback Hell — вложенные колбэки
  - Проблемы с отменой операций
  - Сложная обработка ошибок
- Потоки данных приходят сами — без активного опроса
- События рассматриваются как последовательности данных во времени
- Идеально для:
  - UI-программирования
  - Обработки потоковых данных
  - Асинхронных операций

Почему Rx?

- Библиотека .NET для обработки потоков событий
- Живёт в System.Reactive
- Есть реализации для других языков: RxJava, RxJS, RxSwift
- Расширяет LINQ to Objects на асинхронные последовательности

Основные концепции

- IObservable<T> — представляет наблюдаемую последовательность

```
public interface IObservable<out T>
{
    IDisposable Subscribe(IObserver<T> observer);
}
```

- IObserver<T> — получатель уведомлений, наблюдатель событий

```
public interface IObserver<in T>
{
    void OnNext(T value);       // Новое значение
    void OnError(Exception error); // Ошибка
    void OnCompleted();         // Завершение
}
```

Простой пример

```
// Создаем последовательность
var numbers = Observable.Range(1, 5);

// Подписываемся на нее
numbers.Subscribe(
    x => Console.WriteLine($"Получено: {x}"),
    ex => Console.WriteLine($"Ошибка: {ex.Message}"),
    () => Console.WriteLine("Завершено")
);
Вывод:
Получено: 1
Получено: 2
Получено: 3
Получено: 4
Получено: 5
Завершено
```

Hot/Cold Observables

- Hot — события происходят независимо от подписки
  - Тики таймера, движения мыши и т.д.
  - Подписка не влияет на генерацию событий
- Cold — события генерируются при подписке
  - `Observable.Range`, `Observable.Return`, `ToObservable()` из коллекции
  - Каждый подписчик получает полную последовательность

Создание Observable: фабричные методы

- `Observable.Return(value)` — одно значение, затем завершение
- `Observable.Empty<T>()` — сразу завершается
- `Observable.Never<T>()` — ничего не делает (полезно для таймаутов)
- `Observable.Throw<T>(ex)` — сразу ошибка
- Все они создают cold observable

Observable.Create

- Позволяет реализовать произвольную логику генерации событий
- Поддерживает отмену через возвращаемый IDisposable или Action

```
IObservable<int> Numbers() =>
    Observable.Create<int>(observer => {
        observer.OnNext(1);
        observer.OnNext(2);
        observer.OnCompleted();
        return Disposable.Empty;
    });
```

Observable.Defer

- Создаёт новую последовательность при каждой подписке
- Инициализация происходит только при подписке
- Полезно для ленивой загрузки (файлы, сеть и т.п.)

```
IObservable<string> lines =
    Observable.Defer(() =>
        File.ReadAllLines("log.txt")
            .ToObservable());
```

Генерация последовательностей

- `Observable.Range(start, count)` — целые числа
- `Observable.Generate` — гибкая генерация с состоянием
  - Поддерживает отмену и бесконечные потоки
  - Можно генерировать числа Фибоначчи, простые и др.

Временные последовательности

- `Observable.Interval(period)` — бесконечный поток значений с интервалом
- `Observable.Timer(dueTime)` — одно значение через заданное время
- `Observable.Timer(dueTime, period)` — как `Interval`, но с задержкой старта
- Все операторы используют планировщики

Управление подписками и отмена

- `Subscribe()` возвращает `IDisposable`
- `Dispose()` отменяет подписку и освобождает ресурсы

```
var subscription = numbers.Subscribe(...);
subscription.Dispose(); // отмена
```

- Для множества подписок: `CompositeDisposable`

```
var disposables = new CompositeDisposable();
disposables.Add(obs1.Subscribe(...));
disposables.Add(obs2.Subscribe(...));
// ...
disposables.Dispose(); // отменить всё
```

Обработка ошибок

- Ошибка → `OnError` → последовательность завершена
- Восстановление:
  - `Catch` — заменить ошибку на другую последовательность
  - `Retry` — повторить подписку N раз
  - `RetryWhen` — условный повтор с задержкой

```
Observable.Throw<int>(new Exception("Boom"))
    .Catch(Observable.Return(42))
    .Subscribe(Console.WriteLine); // вывод: 42
```

Операторы

Категория	Примеры операторов
Фильтрация	`Where`, `OfType`, `DistinctUntilChanged`, `Take`, `FirstAsync`
Трансформация	`Select`, `SelectMany`, `Cast`
Агрегация	`Count`, `Sum`, `MinBy`, `Scan`
Разделение	`GroupBy`, `Buffer`, `Window`
Комбинирование	`Merge`, `Concat`, `Zip`, `CombineLatest`, `Switch`
Управление временем	`Throttle`, `Sample`, `Delay`, `Timeout`

Пример: отслеживание изменения статуса судна

```
var statusChanges = receiverHost.Messages
    .GroupBy(m => m.Mmsi)
    .SelectMany(g => g
        .OfType<IAisMessageType1to3>()
        .DistinctUntilChanged(m => m.NavigationStatus)
        .Skip(1));
statusChanges.Subscribe(m =>
    Console.WriteLine($"Судно {m.Mmsi}: {m.NavigationStatus}"));
```

Пример: наблюдение за файлом

```
var lines = Observable
    .FromEventPattern<FileSystemEventArgs>(
        fsw, nameof(fsw.Changed))
    .Where(e => e.EventArgs.Name == "app.log")
    .SelectMany(_ => File.ReadAllLines("app.log")
                         .ToObservable())
    .Where(line => line.Contains("ERROR"))
    .Subscribe(Console.WriteLine);
```

Планировщики

- Управляют контекстом, временем и очередностью выполнения работы
- Отделяют логику от исполнения (аналог `TaskScheduler`)
- Основные встроенные планировщики:
  - `Immediate` — немедленно, в текущем потоке
  - `CurrentThread` — очередь на текущем потоке
  - `TaskPoolScheduler` — через TPL, фоновые потоки
  - `EventLoop` — выделенный однопоточный цикл
  - `DispatcherScheduler` — WPF UI-поток
- Пример: `Observable.Interval(TimeSpan.FromSeconds(1), scheduler)`

Тестирование Rx-кода

- Проблема: операторы вроде `Throttle`, `Timeout`, `Delay` зависят от реального времени
- Решение: виртуальное время через `TestScheduler`
- Пакет: `Microsoft.Reactive.Testing`
- Преимущества:
  - Тесты выполняются мгновенно
  - Поведение детерминировано
  - Нет гонок и «моргающих» тестов
- Ключевые методы:
  - `AdvanceBy(ticks)` — сдвинуть время вперёд
  - `AdvanceTo(time)` — перейти к абсолютному моменту
  - `Start()` — выполнить всё запланированное

Пример: тест таймаута без ожидания

```
var scheduler = new TestScheduler();
var source = Observable.Never<int>();
var timeout = source.Timeout(
    TimeSpan.FromSeconds(1), scheduler);

bool fired = false;
timeout.Subscribe(
    _ => {},
    _ => fired = true);

// Продвигаем виртуальное время на 2 сек
scheduler.AdvanceBy(
    TimeSpan.FromSeconds(2).Ticks);

Assert.IsTrue(fired);
```