Rx.NET

Сазонова Ирина 23.Б10

Реактивное программирование (Reactive Programming)

- **Идея**: системы должны “реагировать” на изменения данных или событий в режиме реального времени
- **Цель**: упростить разработку приложений, работающих с асинхронными операциями
- **Применение**:
  - Обработка пользовательских событий
  - Управление потоками данных (обновления из базы данных)
  - Обработка данных от сенсоров и устройств
  - Мобильная и веб разработка

Основные принципы

- **Потоки данных** – представляют собой последовательность изменений (событий) с течением времени
- **Наблюдатели** – подписываются на потоки данных и реагируют на изменения, выполняя определенные действия
- **Реактивные операторы** – возможность фильтровать, преобразовывать, комбинировать и выполнять другие операции над данными в потоке
- **Асинхронность** – обработка данных без блокировки основного потока

Реактивные расширения (Reactive Extensions)

- **Rx** – библиотека .NET для обработки потоков событий
- `System.Reactive` NuGet package
- Позволяет работать с последовательностями данных так же, как и с коллекциями, используя LINQ-операторы
- **Основные элементы**:
  - Наблюдаемый (Observable) – IObservable<T>
  - Наблюдатель (Observer) – IObserver<T>
  - Операторы – Select, Where, Merge, Throttle
  - Подписка (Subscription)

Интерфейс IObservable<T>

- Это фундаментальная абстракция Rx: последовательность значений типа T (в абстрактном смысле похоже на IEnumerable<T>)

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

- IObservable<T> предоставляет значения, когда они становятся доступными (“вставляет” значения в код, при подписке на IObservable<T>)

Hot and Cold observables

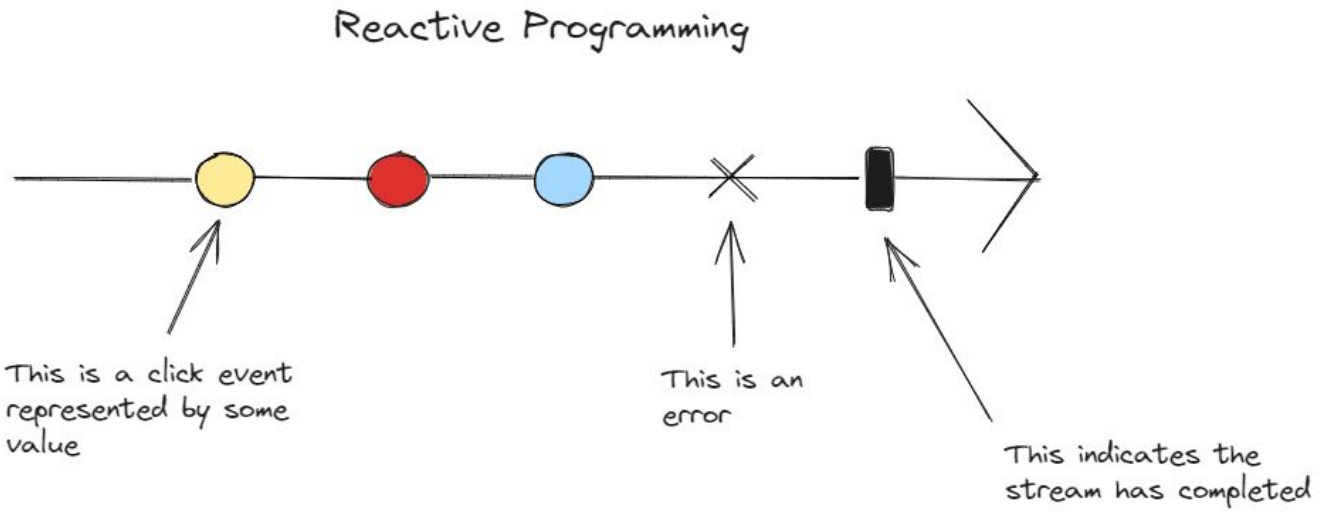
- **Cold Observables**
  - Уведомляют о событиях, когда на них кто-то подписывается
  - Весь поток данных отправляется заново каждому подписчику независимо от времени подписки
  - Данные копируются для каждого подписчика
- **Hot Observables**
  - Пытаются уведомлять о событии независимо от наличия подписчиков. Если на момент события не было подписчиков — данные теряются

Интерфейс IObserver<T>

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

- Удаление подписок (Dispose) необязательно
- Отмена подписок может быть медленной или неэффективной
- Можно подписываться на несколько источников

Интерфейс IObserver<T>



Источник

Реализация IObserver<T>

```
public class MyConsoleObserver<T> : IObserver<T>
{
   public void OnNext(T value)
     => Console.WriteLine($"Received value {value}");

   public void OnError(Exception error)
     => Console.WriteLine($"Sequence failed with {error}");

   public void OnCompleted()
     => Console.WriteLine("Sequence terminated");
}
```

Пример

```
var observable = Observable.Interval(TimeSpan.FromSeconds(1)).Take(5);
// Подписываемся на поток
var subscription = observable.Subscribe(
    onNext: value => Console.WriteLine($"Получено: {value}"),
    onError: error => Console.WriteLine($"Ошибка: {error.Message}"),
    onCompleted: () => Console.WriteLine("Поток завершен.")
);

Console.ReadLine(); // Ждем завершения потока
subscription.Dispose(); // Здесь отписка лишняя
```

```
Получено: 0
Получено: 1
Получено: 2
Получено: 3
Получено: 4
Поток завершен.
```

Реальная задача

Хотим написать программу для мониторинга *температурного датчика*:

1. Каждые 500 миллисекунд происходит генерация случайных данных о температуре
2. Фильтрация значений: учитываются только те, что выше 25 градусов
3. Вычисление скользящего среднего по последним 3 измерениям
4. Реакция на слишком высокую температуру (> 30 градусов) с предупреждением

Генерация температур

```
// Генерация случайных температур каждые 500 мс
var temperatureStream = Observable.Interval(TimeSpan.FromMilliseconds(500))
                                  .Select(_ => GetRandomTemperature());

static double GetRandomTemperature()
{
    var random = new Random();
    return random.NextDouble() * 10 + 20; // Температура от 20°C до 30°
}
```

Where and Buffer

```
// Фильтрация температур выше 25°C
var filteredStream = temperatureStream.Where(temp => temp > 25);

// Вычисление скользящего среднего по последним 3 значениям
var movingAverageStream = filteredStream.Buffer(3, 1)
                                        .Select(buffer => buffer.Average());

// Реакция на слишком высокую температуру (> 30°C)
var highTempWarningStream = filteredStream.Where(temp => temp > 30);
```

Подписка на потоки

```
filteredStream.Subscribe(temp => Console.WriteLine($"Температура: {temp:F2}°C"));

movingAverageStream.Subscribe(avg =>
    Console.WriteLine($"Скользящее среднее: {avg:F2}°C"));

highTempWarningStream.Subscribe(temp =>
    Console.WriteLine($"⚠️ Внимание! Высокая температура: {temp:F2}°C ⚠️"));

Console.WriteLine("Наблюдение за температурой запущено. Нажмите Enter для
завершения.");
Console.ReadLine();
```

Результат

```
Наблюдение за температурой запущено. Нажмите
Enter для завершения.
Температура: 26.5°C
Скользящее среднее: 26.50°C
Температура: 27.2°C
Скользящее среднее: 26.85°C
Температура: 31.0°C
⚠️ Внимание! Высокая температура: 31.0°C ⚠️
Скользящее среднее: 28.23°C
Температура: 28.3°C
Скользящее среднее: 29.50°C
...
```

Почему не .NET Events?

- События требуют ручной реализации фильтрации, буферизации и вычисления среднего
- Ручное управление состоянием (например, придется хранить 3 последние температуры в списке)
- Больше кода
- Сложнее поддерживать асинхронность

Ссылки

- Реактивное программирование
- Введение в реактивное программирование
- Rx.NET