DOTNEXT

DotNext Spb
3 июня 2016
dotnext.ru/spb/

Как приручить реактивное программирование в XAML приложениях

Денис Цветцих
Ice Rock Dev
icerockdev.com

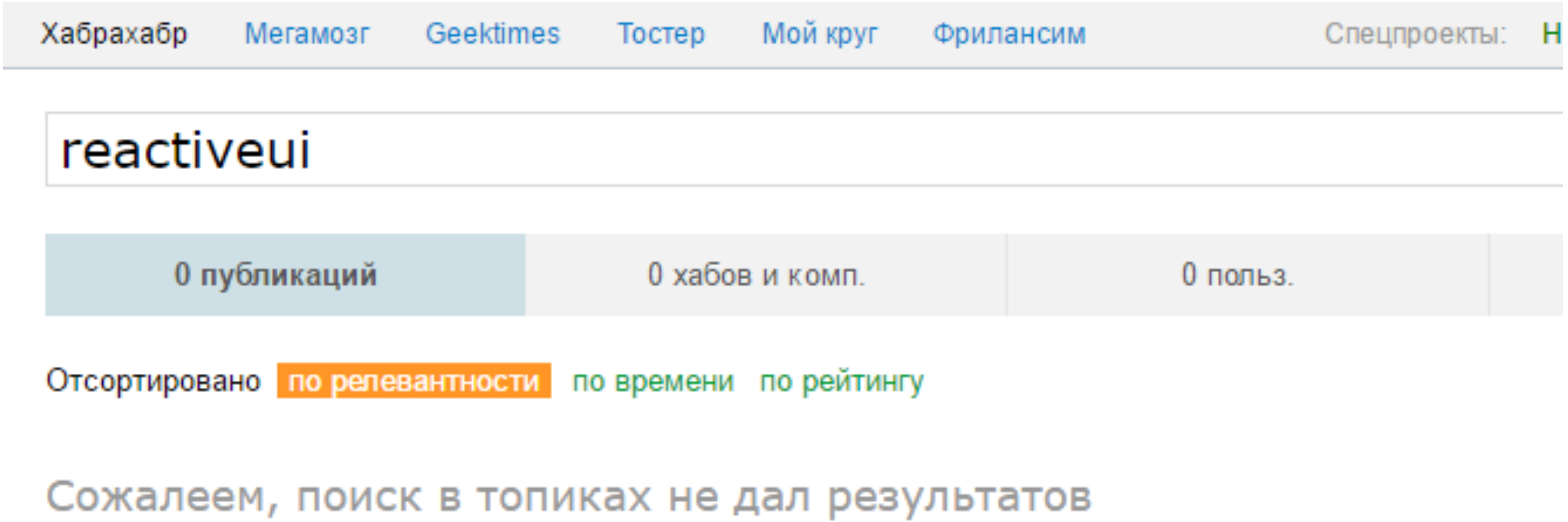
Кто я?

- 7+ лет .NET
- Разработка корпоративных приложений
- Люблю хипстерские библиотеки ☺

Почему я здесь?

- 6 лет назад приручил Rx
- 3 года назад приручил ReactiveUI

ReactiveUI – ни одного поста на хабре



Сэмплы не показательны

Rx – 101 пример задач, где можно обойтись без Rx

ReactiveUI – единственный жизненный сэмпл – поиск (на главной странице)

О чем мы поговорим?

- Что такое реактивное программирование
- Примеры задач из продакшен проектов, которые проще решаются с помощью Rx и ReactiveUI
- Антипаттерны: как ReactiveUI лучше не использовать

ЧТО ТАКОЕ РЕАКТИВНОЕ ПРОГРАММИРОВАНИЕ

Что такое реактивное программирование?

Реактивное программирование – это парадигма программирования, ориентированная на потоки данных и распространение изменений.

(Википедия)

Парадигма программирования

Парадигма программирования – это система идей и понятий, определяющих стиль написания компьютерных программ.

Императивная программа

```
var A = 10;
var B = A + 1;

// Чему равно B?
11

A = A + 1;

// А теперь чему равно B?
11
```

Реактивная программа

```
var A = 10;
var B <- A + 1;
```

<- оператор «судьбы»

```
// Чему равно B?
11

A = A + 1;

// A теперь чему равно B?
12
```

Реактивное программирование

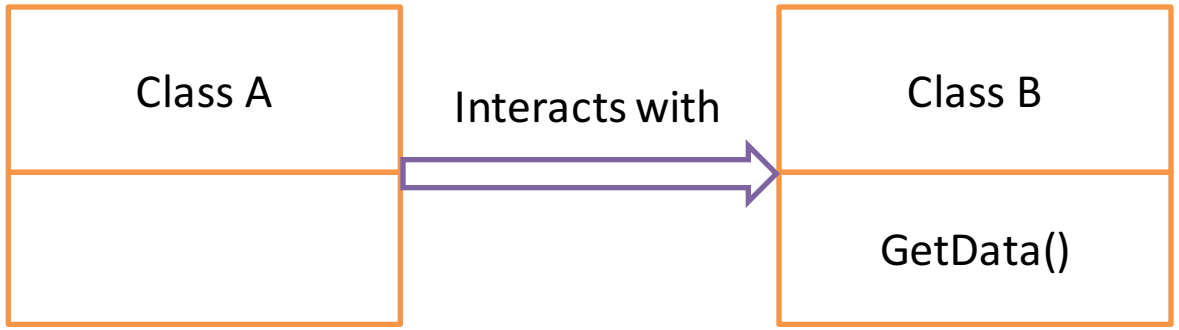
Реактивное программирование – это парадигма программирования, ориентированная на потоки данных и распространение изменений.

Поток данных – последовательность значений переменной или свойства класса.

Распространение изменений – уведомления «заинтересованных» об изменениях.

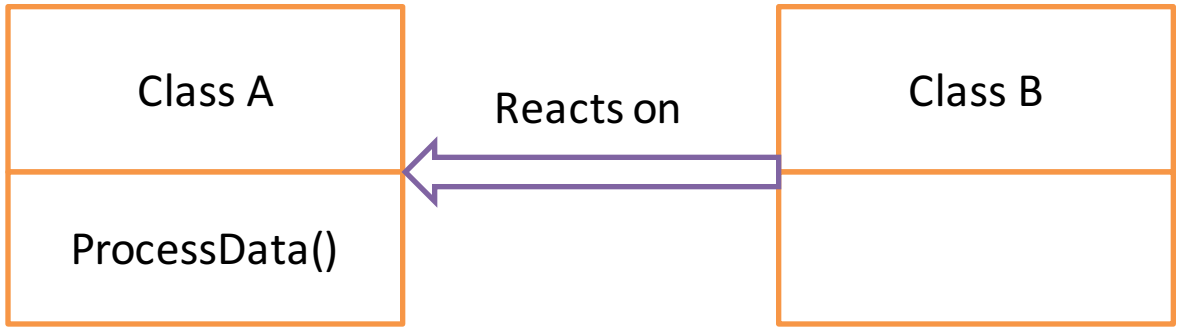
Pull

Pull – класс A взаимодействует с классом B и вытягивает из него необходимые данные



Push

Push – класс B самостоятельно выталкивает данные классу A, как только они становятся доступны.



РЕАКТИВНОЕ ПРОГРАММИРОВАНИЕ НА C#

Реактивное программирование на .NET

- Reactive Extensions (Rx)
  - Эрик Мейер из MS Research
- ReactiveUI
  - Разработана в MS Research
  - библиотека на базе Rx для создания элегантных UI для всех XAML платформ

Внутри .NET

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

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

Внутри .NET

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

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

Внутри .NET

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

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

Внутри .NET

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

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

Внутри .NET

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

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

Оператор судьбы через Rx

```
var A = new Subject<int>();

// B <- A + 1
var B = A.Select(a => a + 1);

A.OnNext(10);
```

Оператор судьбы через Rx

```
var A = new Subject<int>();

// B <- A + 1
var B = A.Select(a => a + 1);

A.OnNext(10);
```

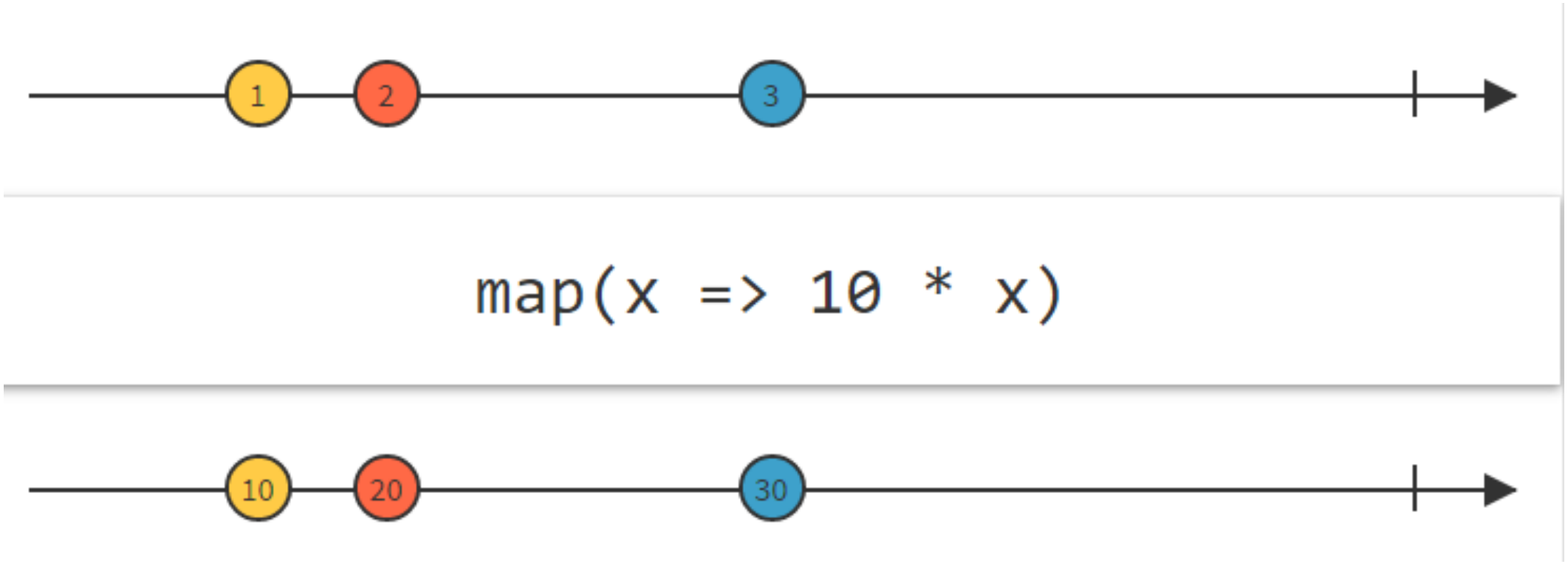
Оператор судьбы через Rx

```
var A = new Subject<int>();

// B <- A + 1
var B = A.Select(a => a + 1);

A.OnNext(10);
```

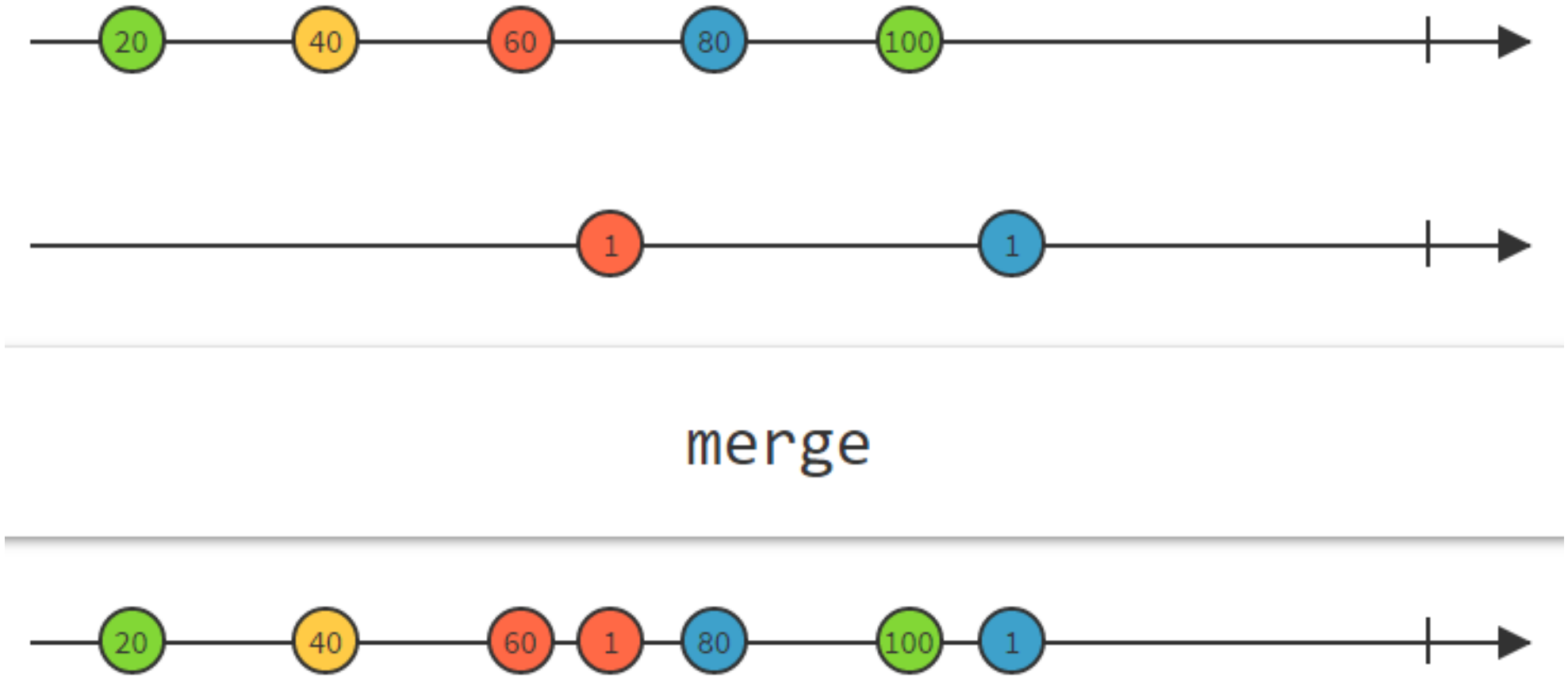
Оператор судьбы через Rx

```
var A = new Subject<int>();

// B <- A + 1
var B = A.Select(a => a + 1);

A.OnNext(10);
```

Observable.Select – проекция

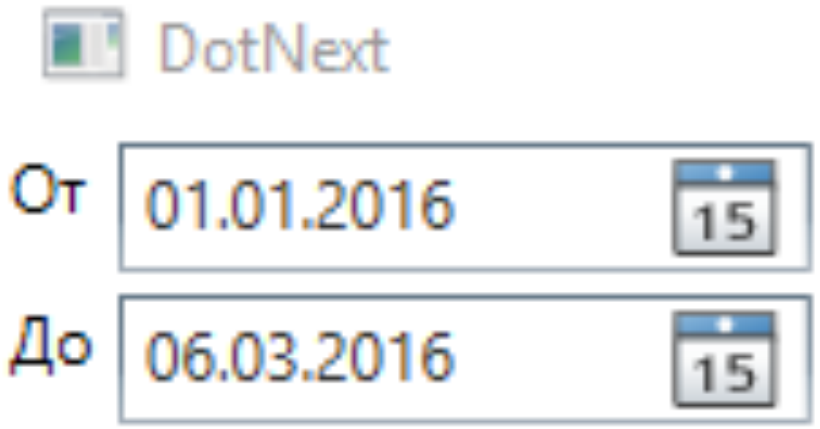


Observable.Merge – слияние



КАКИЕ ЗАДАЧИ МОЖНО РЕШАТЬ

Пример



Приложение для поиска по материалам конференции DotNext

Для читабельности примеров

```
public class Filter : ReactiveObject
{
    private DateTime _fromDate;
    private DateTime _toDate;

    public DateTime FromDate
    {
        get { return _fromDate; }
        set { this.RaiseAndSetIfChanged(ref _fromDate, value); }
    }

    public DateTime ToDate
    {
        get { return _toDate; }
        set { this.RaiseAndSetIfChanged(ref _toDate, value); }
    }
}
```

Для читабельности примеров

```
public class Filter : ReactiveObject
{
    private DateTime _fromDate;
    private DateTime _toDate;

    public DateTime FromDate
    {
        get { return _fromDate; }
        set { this.RaiseAndSetIfChanged(ref _fromDate, value); }
    }

    public DateTime ToDate
    {
        get { return _toDate; }
        set { this.RaiseAndSetIfChanged(ref _toDate, value); }
    }
}
```

Для читабельности примеров

```
public class Filter : ReactiveObject
{
    private DateTime _fromDate;
    private DateTime _toDate;

    public DateTime FromDate
    {
        get { return _fromDate; }
        set { this.RaiseAndSetIfChanged(ref _fromDate, value); }
    }

    public DateTime ToDate
    {
        get { return _toDate; }
        set { this.RaiseAndSetIfChanged(ref _toDate, value); }
    }
}
```

Для читабельности примеров

```
public class Filter
{
    public DateTime FromDate { get; set; }

    public DateTime ToDate { get; set; }
}
```

Задача 1: подписка на события об изменении свойств фильтра

Обычная подписка на событие PropertyChanged

```
var filter = new Filter();

// Подписка
filter.PropertyChanged += OnPropertyChanged;


// Реакция на событие
private void OnPropertyChanged
             (object sender, PropertyChangedEventArgs e)
{

}

// Отписка
filter.PropertyChanged -= OnPropertyChanged;
```

Обычная подписка на событие PropertyChanged

```
var filter = new Filter();

// Подписка
filter.PropertyChanged += OnPropertyChanged;


// Реакция на событие
private void OnPropertyChanged
             (object sender, PropertyChangedEventArgs e)
{

}

// Отписка
filter.PropertyChanged -= OnPropertyChanged;
```

Обычная подписка на событие PropertyChanged

```
var filter = new Filter();

// Подписка
filter.PropertyChanged += OnPropertyChanged;


// Реакция на событие
private void OnPropertyChanged
             (object sender, PropertyChangedEventArgs e)
{

}

// Отписка
filter.PropertyChanged -= OnPropertyChanged;
```

Обычная подписка на событие PropertyChanged

```
var filter = new Filter();

// Подписка
filter.PropertyChanged += OnPropertyChanged;


// Реакция на событие
private void OnPropertyChanged
             (object sender, PropertyChangedEventArgs e)
{

}

// Отписка
filter.PropertyChanged -= OnPropertyChanged;
```

Rx подписка на событие PropertyChanged

```
filter = new Filter();

// Подписка
IDisposable subscription =
    Observable.FromEventPattern<PropertyChangedEventArgs>
    (filter, "PropertyChanged")
    .Subscribe(OnPropertyChanged);

// Реакция на событие
private void OnPropertyChanged
    (EventPattern<PropertyChangedEventArgs> eventPattern)
{
     //eventPattern.Sender и eventPattern.EventArgs
}

// Отписка
subscription.Dispose();
```

Rx подписка на событие PropertyChanged

```
filter = new Filter();

// Подписка
IDisposable subscription =
    Observable.FromEventPattern<PropertyChangedEventArgs>
    (filter, "PropertyChanged")
    .Subscribe(OnPropertyChanged);

// Реакция на событие
private void OnPropertyChanged
    (EventPattern<PropertyChangedEventArgs> eventPattern)
{
     //eventPattern.Sender и eventPattern.EventArgs
}

// Отписка
subscription.Dispose();
```

Rx подписка на событие PropertyChanged

```
filter = new Filter();

// Подписка
IDisposable subscription =
    Observable.FromEventPattern<PropertyChangedEventArgs>
    (filter, "PropertyChanged")
    .Subscribe(OnPropertyChanged);

// Реакция на событие
private void OnPropertyChanged
    (EventPattern<PropertyChangedEventArgs> eventPattern)
{
     //eventPattern.Sender и eventPattern.EventArgs
}

// Отписка
subscription.Dispose();
```

Rx подписка на событие PropertyChanged

```
filter = new Filter();

// Подписка
IDisposable subscription =
    Observable.FromEventPattern<PropertyChangedEventArgs>
    (filter, "PropertyChanged")
    .Subscribe(OnPropertyChanged);

// Реакция на событие
private void OnPropertyChanged
    (EventPattern<PropertyChangedEventArgs> eventPattern)
{
     //eventPattern.Sender и eventPattern.EventArgs
}

// Отписка
subscription.Dispose();
```

Rx подписка на событие PropertyChanged

```
filter = new Filter();

// Подписка
IDisposable subscription =
    Observable.FromEventPattern<PropertyChangedEventArgs>
    (filter, "PropertyChanged")
    .Subscribe(OnPropertyChanged);

// Реакция на событие
private void OnPropertyChanged
    (EventPattern<PropertyChangedEventArgs> eventPattern)
{
     //eventPattern.Sender и eventPattern.EventArgs
}

// Отписка
subscription.Dispose();
```

Rx подписка на событие PropertyChanged

```
filter = new Filter();

// Подписка
IDisposable subscription =
    Observable.FromEventPattern<PropertyChangedEventArgs>
    (filter, "PropertyChanged")
    .Subscribe(OnPropertyChanged);

// Реакция на событие
private void OnPropertyChanged
    (EventPattern<PropertyChangedEventArgs> eventPattern)
{
     //eventPattern.Sender и eventPattern.EventArgs
}

// Отписка
subscription.Dispose();
```

Достоинства Rx подписки

- Подписка инкапсулируется в IDisposable объекте, просто сделать отписку
- Подписка – это поток данных, над которым можно выполнять операции

Задача 2: подписка на изменение свойства FromDate

Обычная подписка на изменение свойства FromDate

```
Filter filter = new Filter();

filter.PropertyChanged += OnPropertyChanged;

private void OnPropertyChanged
    (object sender, PropertyChangedEventArgs eventArgs)
{
    if (eventArgs.PropertyName == “FromDate")
    {
        // Действия
    }
}
```

Обычная подписка на изменение свойства FromDate

```
Filter filter = new Filter();

filter.PropertyChanged += OnPropertyChanged;

private void OnPropertyChanged
    (object sender, PropertyChangedEventArgs eventArgs)
{
    if (eventArgs.PropertyName == “FromDate")
    {
        // Действия
    }
}
```

Обычная подписка на изменение свойства FromDate

```
Filter filter = new Filter();

filter.PropertyChanged += OnPropertyChanged;

private void OnPropertyChanged
    (object sender, PropertyChangedEventArgs eventArgs)
{
    if (eventArgs.PropertyName == “FromDate")
    {
        // Действия
    }
}
```

ReactiveUI подписка на изменение свойства FromDate

```
var filter = new Filter();

var subscription = filter
    .ObservableForProperty(f => f.FromDate)
    .Subscribe(FromDateChanged);

private void FromDateChanged
    (IObservedChange<Filter, DateTime> change)
{
    // Действия
}
```

ReactiveUI подписка на изменение свойства FromDate

```
var filter = new Filter();

var subscription = filter
    .ObservableForProperty(f => f.FromDate)
    .Subscribe(FromDateChanged);

private void FromDateChanged
    (IObservedChange<Filter, DateTime> change)
{
    // Действия
}
```

ReactiveUI подписка на изменение свойства FromDate

```
var filter = new Filter();

var subscription = filter
    .ObservableForProperty(f => f.FromDate)
    .Subscribe(FromDateChanged);

private void FromDateChanged
    (IObservedChange<Filter, DateTime> change)
{
    // Действия
}
```

ReactiveUI подписка на изменение свойства FromDate

```
var filter = new Filter();

var subscription = filter
    .ObservableForProperty(f => f.FromDate)
    .Subscribe(FromDateChanged);
```

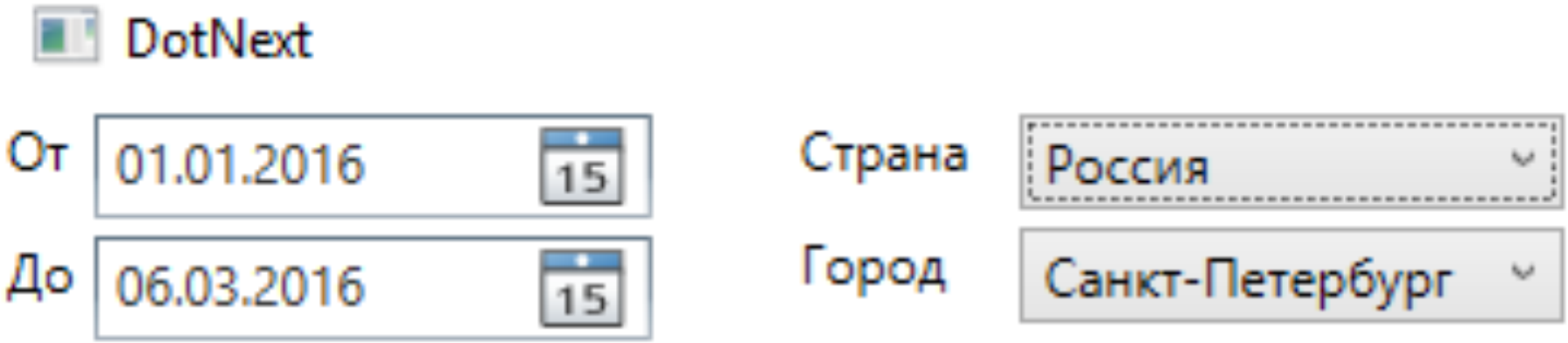
```
private void FromDateChanged
    (IObservedChange<Filter, DateTime> change)
{
    // Действия
}
```

Достоинства ObservableForProperty

- Не нужно проверять, что обрабатываем событие об изменении нужного свойства

Задача 3: действия при изменении значений свойств

Добавим выбор страны и города



Добавим в фильтр:

- Список стран и городов
- При выборе страны актуализируем список городов

Валидация выбранного периода

Условие: FromDate не превосходит ToDate

Если FromDate > ToDate, тогда ToDate приравниваем к FromDate

Если ToDate < FromDate, тогда FromDate приравниваем к ToDate

Обычное решение

```
private DateTime _fromDate;

public DateTime FromDate
{
    get { return _fromDate; }
    set
    {
        _fromDate = value;
        OnPropertyChanged();

        if (_fromDate > ToDate) ToDate = _fromDate;
    }
}
```

Обычное решение

```
private DateTime _fromDate;

public DateTime FromDate
{
    get { return _fromDate; }
    set
    {
        _fromDate = value;
        OnPropertyChanged();

        if (_fromDate > ToDate) ToDate = _fromDate;
    }
}
```

ObservableForProperty

```
private IDisposable _subscription;

public Filter()
{
    _subscription =
        this.ObservableForProperty(vm => vm.FromDate)
      .Where(fromDate => fromDate.Value > ToDate)
      .Subscribe(fromDate => ToDate = fromDate.Value);
}

public DateTime FromDate { get; set; } // PropertyChanged
```

ObservableForProperty

```
private IDisposable _subscription;

public Filter()
{
    _subscription =
        this.ObservableForProperty(vm => vm.FromDate)
      .Where(fromDate => fromDate.Value > ToDate)
      .Subscribe(fromDate => ToDate = fromDate.Value);
}

public DateTime FromDate { get; set; } // PropertyChanged
```

ObservableForProperty

```
private IDisposable _subscription;

public Filter()
{
    _subscription =
        this.ObservableForProperty(vm => vm.FromDate)
      .Where(fromDate => fromDate.Value > ToDate)
      .Subscribe(fromDate => ToDate = fromDate.Value);
}

public DateTime FromDate { get; set; } // PropertyChanged
```

ObservableForProperty

```
private IDisposable _subscription;

public Filter()
{
    _subscription =
        this.ObservableForProperty(vm => vm.FromDate)
      .Where(fromDate => fromDate.Value > ToDate)
      .Subscribe(fromDate => ToDate = fromDate.Value);
}

public DateTime FromDate { get; set; } // PropertyChanged
```

ObservableForProperty

```
private IDisposable _subscription;

public Filter()
{
    _subscription =
        this.ObservableForProperty(vm => vm.FromDate)
      .Where(fromDate => fromDate.Value > ToDate)
      .Subscribe(fromDate => ToDate = fromDate.Value);
}

public DateTime FromDate { get; set; } // PropertyChanged

if (_fromDate > ToDate) ToDate = _fromDate;
```

Зависимости между свойствами в обычном решении

```
public class Filter
{

    public DateTime FromDate
    {
        get { return _fromDate; }
        set
        {
            _fromDate = value;
            if (_fromDate > ToDate) ToDate = FromDate;
        }
    }


    public DateTime ToDate
    {
        get { return _toDate; }
        set
        {
            _toDate = value;
            if (_toDate < FromDate) FromDate = _toDate;
        }
    }

    public List<string> Cities
    {
        get { return _cities; }
        set
        {
            _cities = value;
            SelectedCity = null;
        }
    }

    public string SelectedCountry
    {
        get { return _selectedCountry; }
        set
        {
            _selectedCountry = value;
            Cities = GetCitiesForCountry(_selectedCountry);
        }
    }

}
```

Зависимости между свойствами в ReactiveUI решении

```
public class Filter
{
    public Filter()
    {
        _fromDateSubscription =
            this.ObservableForProperty(filter => filter.FromDate)
            .Where(fromDate => fromDate.Value > ToDate)
            .Subscribe(fromDate => ToDate = fromDate.Value);

        _toDateSubscription =
            this.ObservableForProperty(filter => filter.ToDate)
            .Where(toDate => toDate.Value < FromDate)
            .Subscribe(toDate => FromDate = toDate.Value);

        _selectedCountrySubscription =
            this.ObservableForProperty(filter => filter.SelectedCountry)
            .Subscribe(country => Cities = GetCitesForCountry(country.Value));

        _citiesSubscription = this.ObservableForProperty(filter => Cities)
            .Subscribe(_ => SelectedCity = null);
    }

    public DateTime FromDate { get; set; }

    public DateTime ToDate { get; set; }

    public List<string> Countries { get; set; }

    public List<string> Cities { get; set; }

    public string SelectedCountry { get; set; }

    public string SelectedCity { get; set; }

}
```

Достоинства ObservableForProperty

- Удаление логики из сеттеров
- Делает код на порядок более читаемым, собирая код зависимости между свойствами в конструкторе

Задача 4: мониторинг одновременно нескольких свойств

Поля фильтра:

- От
- До
- Страна
- Город

При изменении любого – обновляем данные

ViewModel с фильтром

```
public class ViewModel
{
    public Filter Filter { get; set; }

    public ViewModel()
    {
        Filter = new Filter();

        // Подписка на изменение свойств фильтра


    }
}
```

ViewModel с фильтром

```
public class ViewModel
{
    public Filter Filter { get; set; }

    public ViewModel()
    {
        Filter = new Filter();

        // Подписка на изменение свойств фильтра


    }
}
```

ViewModel с фильтром

```
public class ViewModel
{
    public Filter Filter { get; set; }

    public ViewModel()
    {
        Filter = new Filter();

        // Подписка на изменение свойств фильтра

    }
}
```

Обычное решение

```
public ViewModel()
{
    Filter = new Filter();

    // Подписка на изменение свойств фильтра
    Filter.PropertyChanged += OnFilterPropertyChanged;
}


private void OnFilterPropertyChanged
    (object sender, PropertyChangedEventArgs eventArgs)
{
    if (eventArgs.PropertyName == "FromDate" ||
        eventArgs.PropertyName == "ToDate" ||
        eventArgs.PropertyName == "SelectedCountry" ||
        eventArgs.PropertyName == "SelectedCity")
    {
        Refresh();
    }
}
```

Обычное решение

```
public ViewModel()
{
    Filter = new Filter();

    // Подписка на изменение свойств фильтра
    Filter.PropertyChanged += OnFilterPropertyChanged;
}


private void OnFilterPropertyChanged
    (object sender, PropertyChangedEventArgs eventArgs)
{
    if (eventArgs.PropertyName == "FromDate" ||
        eventArgs.PropertyName == "ToDate" ||
        eventArgs.PropertyName == "SelectedCountry" ||
        eventArgs.PropertyName == "SelectedCity")
    {
        Refresh();
    }
}
```

ReactiveUI решение

```
public ViewModel()
{
    Filter = new Filter();

   // Подписка на изменение свойств фильтра

   _subscription =
       Filter.WhenAnyValue(
            f => f.FromDate,
            f => f.ToDate,
            f => f.SelectedCountry,
            f => f.SelectedCity)
        .Subscribe(_ => Refresh());
}
```

ReactiveUI решение

```
public ViewModel()
{
    Filter = new Filter();

   // Подписка на изменение свойств фильтра

   _subscription =
       Filter.WhenAnyValue(
            f => f.FromDate,
            f => f.ToDate,
            f => f.SelectedCountry,
            f => f.SelectedCity)
        .Subscribe(_ => Refresh());
}
```

ReactiveUI решение

```
public ViewModel()
{
    Filter = new Filter();

   // Подписка на изменение свойств фильтра

   _subscription =
       Filter.WhenAnyValue(
            f => f.FromDate,
            f => f.ToDate,
            f => f.SelectedCountry,
            f => f.SelectedCity)
        .Subscribe(_ => Refresh());
}
```

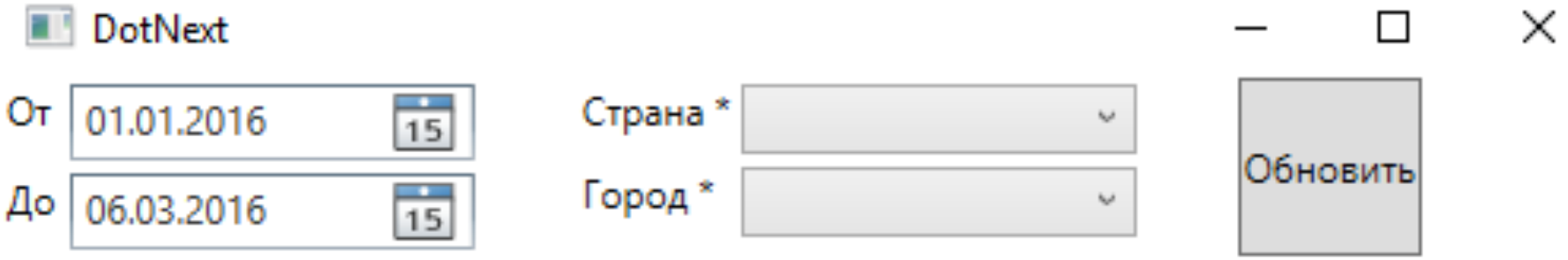
ReactiveUI решение

```
public ViewModel()
{
    Filter = new Filter();

   // Подписка на изменение свойств фильтра

   _subscription =
       Filter.WhenAnyValue(
            f => f.FromDate,
            f => f.ToDate,
            f => f.SelectedCountry,
            f => f.SelectedCity)
        .Subscribe(_ => Refresh());
}
```

Достоинства WhenAnyValue

- Подписываемся только на нужные нам свойства
- Можно подписываться на изменения вложенных свойств
  - Obj.Deep1.Deep2.Deep3…

Задача 5: проверка CanExecute команды



- Добавляем кнопку «Обновить» на фильтр
- Страна и город – обязательные поля

RelayCommand из MvvmLight

```
public RelayCommand RefreshCommand { get; private set; }

public Filter()
{
    RefreshCommand = new RelayCommand(
        OnRefresh,
        () => SelectedCity != null && SelectedCountry != null);
}
```

RelayCommand из MvvmLight

```
public RelayCommand RefreshCommand { get; private set; }

public Filter()
{
    RefreshCommand = new RelayCommand(
        OnRefresh,
        () => SelectedCity != null && SelectedCountry != null);
}
```

RelayCommand из MvvmLight

```
public RelayCommand RefreshCommand { get; private set; }

public Filter()
{
    RefreshCommand = new RelayCommand(
        OnRefresh,
        () => SelectedCity != null && SelectedCountry != null);
}
```

RelayCommand из MvvmLight

```
public RelayCommand RefreshCommand { get; private set; }

public Filter()
{
    RefreshCommand = new RelayCommand(
        OnRefresh,
        () => SelectedCity != null && SelectedCountry != null);
}
```

Подергивания ☺

```
public string SelectedCountry {
    get { return _selectedCountry; }
    set {
        _selectedCountry = value; OnPropertyChanged();

        RefreshCommand.RaiseCanExecuteChanged();
    }
}

public string SelectedCity {
    get { return _selectedCity; }
    set {
        _selectedCity = value; OnPropertyChanged();

        RefreshCommand.RaiseCanExecuteChanged();
    }
}
```

ReactiveCommand

```
public Filter()
{
    var canExecute = this.WhenAny(
       f => f.SelectedCity,
       f => f.SelectedCountry,
       (city, country) =>
              city.Value != null && country.Value != null);

    var command = ReactiveCommand.Create(canExecute);
    command.Subscribe(_ => OnRefresh());

    RefreshCommand = command;
}

public string SelectedCountry { get; set; }
```

ReactiveCommand

```
public Filter()
{
    var canExecute = this.WhenAny(
       f => f.SelectedCity,
       f => f.SelectedCountry,
       (city, country) =>
              city.Value != null && country.Value != null);

    var command = ReactiveCommand.Create(canExecute);
    command.Subscribe(_ => OnRefresh());

    RefreshCommand = command;
}

public string SelectedCountry { get; set; }
```

ReactiveCommand

```
public Filter()
{
    var canExecute = this.WhenAny(
      f => f.SelectedCity,
      f => f.SelectedCountry,
      (city, country) =>
            city.Value != null && country.Value != null);

    var command = ReactiveCommand.Create(canExecute);
    command.Subscribe(_ => OnRefresh());

    RefreshCommand = command;
}

public string SelectedCountry { get; set; }
```

ReactiveCommand

```
public Filter()
{
    var canExecute = this.WhenAny(
       f => f.SelectedCity,
       f => f.SelectedCountry,
       (city, country) =>
             city.Value != null && country.Value != null);

    var command = ReactiveCommand.Create(canExecute);
    command.Subscribe(_ => OnRefresh());

    RefreshCommand = command;
}

public string SelectedCountry { get; set; }
```

ReactiveCommand

```
public Filter()
{
    var canExecute = this.WhenAny(
       f => f.SelectedCity,
       f => f.SelectedCountry,
       (city, country) =>
              city.Value != null && country.Value != null);

    var command = ReactiveCommand.Create(canExecute);
    command.Subscribe(_ => OnRefresh());

    RefreshCommand = command;
}

public string SelectedCountry { get; set; }
```

ReactiveCommand

```
public Filter()
{
    var canExecute = this.WhenAny(
       f => f.SelectedCity,
       f => f.SelectedCountry,
       (city, country) =>
              city.Value != null && country.Value != null);

    var command = ReactiveCommand.Create(canExecute);
    command.Subscribe(  => OnRefresh());

    RefreshCommand = command;
}

public string SelectedCountry { get; set; }
// RefreshCommand.RaiseCanExecuteChanged();
```

Достоинства ReactiveCommand

- Не нужны *«подергивания»* в сеттерах
- При изменения условия нельзя забыть убрать или добавить «*подергивание*»

Задача 6: команда запускает асинхронное действие

- При выполнении команды – вызов вебсервиса
- Пока не получим ответ от сервиса – команду нельзя выполнить

Асинхронное действие RelayCommand

```
public bool IsBusy  { get; set; }
//RefreshCommand.RaiseCanExecuteChanged();

public Filter(Task refreshTask)
{
    RefreshTask = refreshTask;

    RefreshCommand = new RelayCommand(OnRefresh,
        () => SelectedCity != null &&
              SelectedCountry != null &&
              !IsBusy);
}
```

Асинхронное действие RelayCommand

```
public bool IsBusy  { get; set; }
//RefreshCommand.RaiseCanExecuteChanged();

public Filter(Task refreshTask)
{
    RefreshTask = refreshTask;

    RefreshCommand = new RelayCommand(OnRefresh,
        () => SelectedCity != null &&
              SelectedCountry != null &&
              !IsBusy);
}
```

Асинхронное действие RelayCommand

```
public bool IsBusy  { get; set; }
//RefreshCommand.RaiseCanExecuteChanged();

public Filter(Task refreshTask)
{
    RefreshTask = refreshTask;

    RefreshCommand = new RelayCommand(OnRefresh,
        () => SelectedCity != null &&
              SelectedCountry != null &&
              !IsBusy);
}
```

Асинхронное обновление

```
private async void OnRefresh()
{
    IsBusy = true;

    await RefreshTask; // запрос на сервер

    IsBusy = false;
}
```

Асинхронное обновление

```
private async void OnRefresh()
{
    IsBusy = true;

    await RefreshTask; // запрос на сервер

    IsBusy = false;
}
```

Асинхронное обновление

```
private async void OnRefresh()
{
    IsBusy = true;

    await RefreshTask; // запрос на сервер

    IsBusy = false;
}
```

ReactiveAsyncCommand

```
public Filter(Func<object, Task> refreshFunc)
{
    var canExecute = this.WhenAny(
        f => f.SelectedCity, f => f.SelectedCountry,
        (city, country) =>
          city.Value != null && country.Value != null);

    var command = ReactiveCommand.CreateAsyncTask(
        canExecute,
        refreshFunc);

    RefreshCommand = command;
}
```

ReactiveAsyncCommand

```
public Filter(Func<object, Task> refreshFunc)
{
    var canExecute = this.WhenAny(
        f => f.SelectedCity, f => f.SelectedCountry,
        (city, country) =>
          city.Value != null && country.Value != null);

    var command = ReactiveCommand.CreateAsyncTask(
        canExecute,
        refreshFunc);

    RefreshCommand = command;
}
```

ReactiveAsyncCommand

```
public Filter(Func<object, Task> refreshFunc)
{
    var canExecute = this.WhenAny(
        f => f.SelectedCity, f => f.SelectedCountry,
        (city, country) =>
          city.Value != null && country.Value != null);

    var command = ReactiveCommand.CreateAsyncTask(
        canExecute,
        refreshFunc);

    RefreshCommand = command;
}
```

ReactiveAsyncCommand

```
public Filter(Func<object, Task> refreshFunc)
{
    var canExecute = this.WhenAny(
        f => f.SelectedCity, f => f.SelectedCountry,
        (city, country) =>
          city.Value != null && country.Value != null);

    var command = ReactiveCommand.CreateAsyncTask(
        canExecute,
        refreshFunc);

    RefreshCommand = command;
}
```

ReactiveAsyncCommand

```
public Filter(Func<object, Task> refreshFunc)
{
    var canExecute = this.WhenAny(
        f => f.SelectedCity, f => f.SelectedCountry,
        (city, country) =>
          city.Value != null && country.Value != null);

    var command = ReactiveCommand.CreateAsyncTask(
        canExecute,
        refreshFunc);

    RefreshCommand = command;
}
```

Демо

Достоинства ReactiveAsyncCommand

- Не нужно отдельного свойства для мониторинга начала/конца операции

Задача 7: поиск по подстроке

Вместо фильтра делаем поиск

Если длина строки < 3, то ждем 0,5 сек перед поиском

Если длина строки >= 3, то ждем 0,25 сек перед поиском

Поиск по строке

```
public class ViewModel
{
    public Filter Filter { get; set; }

    public string SearchText { get; set; }
}
```

Поиск по строке

```
public class ViewModel
{
    public Filter Filter { get; set; }

    public string SearchText { get; set; }
}
```

Поиск по строке

```
public class ViewModel
{
    public Filter Filter { get; set; }

    public string SearchText { get; set; }
}
```

Where, Throttle

```
var textObservable =
    this.ObservableForProperty(s => s.SearchText);

var firstRule = textObservable
    .Where(text => text.Value == null || text.Value.Length < 3)
    .Throttle(TimeSpan.FromMilliseconds(500));

var secondRule = textObservable
    .Where(text => text.Value != null && text.Value.Length >= 3)
    .Throttle(TimeSpan.FromMilliseconds(250));
```

Where, Throttle

```
var textObservable =
    this.ObservableForProperty(s => s.SearchText);

var firstRule = textObservable
    .Where(text => text.Value == null || text.Value.Length < 3)
    .Throttle(TimeSpan.FromMilliseconds(500));

var secondRule = textObservable
    .Where(text => text.Value != null && text.Value.Length >= 3)
    .Throttle(TimeSpan.FromMilliseconds(250));
```

Where, Throttle

```
var textObservable =
    this.ObservableForProperty(s => s.SearchText);

var firstRule = textObservable
    .Where(text => text.Value == null || text.Value.Length < 3)
    .Throttle(TimeSpan.FromMilliseconds(500));

var secondRule = textObservable
    .Where(text => text.Value != null && text.Value.Length >= 3)
    .Throttle(TimeSpan.FromMilliseconds(250));
```

Where, Throttle

```
var textObservable =
    this.ObservableForProperty(s => s.SearchText);

var firstRule = textObservable
    .Where(text => text.Value == null || text.Value.Length < 3)
    .Throttle(TimeSpan.FromMilliseconds(500));

var secondRule = textObservable
    .Where(text => text.Value != null && text.Value.Length >= 3)
    .Throttle(TimeSpan.FromMilliseconds(250));
```

Where, Throttle

```
var textObservable =
    this.ObservableForProperty(s => s.SearchText);

var firstRule = textObservable
    .Where(text => text.Value == null || text.Value.Length < 3)
    .Throttle(TimeSpan.FromMilliseconds(500));

var secondRule = textObservable
    .Where(text => text.Value != null && text.Value.Length >= 3)
    .Throttle(TimeSpan.FromMilliseconds(250));
```

Where, Throttle

```
var textObservable =
    this.ObservableForProperty(s => s.SearchText);

var firstRule = textObservable
    .Where(text => text.Value == null || text.Value.Length < 3)
    .Throttle(TimeSpan.FromMilliseconds(500));

var secondRule = textObservable
    .Where(text => text.Value != null && text.Value.Length >= 3)
    .Throttle(TimeSpan.FromMilliseconds(250));
```

Merge, ObserveOnDispatcher

```
var resultObservable =
  firstRule.Merge(secondRule);

resultObservable
    .ObserveOnDispatcher()
    .Subscribe(Refresh);
```

Merge, ObserveOnDispatcher

```
var resultObservable =
  firstRule.Merge(secondRule);


resultObservable
    .ObserveOnDispatcher()
    .Subscribe(Refresh);
```

Merge, ObserveOnDispatcher

```
var resultObservable =
  firstRule.Merge(secondRule);


resultObservable
    .ObserveOnDispatcher()
    .Subscribe(Refresh);
```

Достоинства

- На порядок меньшее кода, чем написали бы для обычного решения
- Код простой и читаемый

АНТИПАТТЕРН: REACTIVEUI КАК MVVM

View – ViewModel маппинг

```
private void RegisterParts(IMutableDependencyResolver dependencyResolver)
{
    dependencyResolver.RegisterConstant(this, typeof(IScreen));

    dependencyResolver.Register(
        () => new WelcomeView(), typeof(IViewFor<WelcomeViewModel>));
}
```

View – ViewModel маппинг

```
private void RegisterParts(IMutableDependencyResolver dependencyResolver)
{
    dependencyResolver.RegisterConstant(this, typeof(IScreen));

    dependencyResolver.Register(
        () => new WelcomeView(), typeof(IViewFor<WelcomeViewModel>));
}
```

ViewModel First навигация

```
var viewModel =
    (ViewModel) Locator.CurrentMutable.GetService(typeof(ViewModel));
viewModel.Title = "New ViewModel";
Router.Navigate.Execute(viewModel);
```

ViewModel First навигация

```
var viewModel =
    (ViewModel) Locator.CurrentMutable.GetService(typeof(ViewModel));
viewModel.Title = "New ViewModel";
Router.Navigate.Execute(viewModel);
```

ViewModel First навигация

```
var viewModel =
    (ViewModel) Locator.CurrentMutable.GetService(typeof(ViewModel));
viewModel.Title = "New ViewModel";
Router.Navigate.Execute(viewModel);
```

ViewModel First навигация

```
var viewModel =
    (ViewModel) Locator.CurrentMutable.GetService(typeof(ViewModel));
viewModel.Title = "New ViewModel";
Router.Navigate.Execute(viewModel);
```

Нет CompositeUI

```
<Grid UseLayoutRounding="True">
    <!--
        COOLSTUFF: RoutedViewHost

        RoutedViewHost will take a Router and display whatever ViewModel
        is at the front of the route stack.
    -->
    <rx:RoutedViewHost
        Router="{Binding Router}"
        HorizontalContentAlignment="Stretch"
        VerticalContentAlignment="Stretch" />
</Grid>
```

Нельзя избавиться от Splat IoC

В Splat

- Получить экземпляр сервиса – аналог ServiceLocator

```
Locator.CurrentMutable.GetService(typeof(Service))
```

Выход

- Реализовать IMutableDependencyResolver (для AutoFac)

Недостатки ReactiveUI как MVVM

- Нельзя задать конвенцию именования для маппинга View-ViewModel
- Нет API, упрощающего задание соответствия View-ViewModel
- Splat – это ServiceLocator в чистом виде
- Нельзя заменить Splat на свой любимый IoC
- Нельзя реализовать CompositeUI
- ViewModelFirst навигация неудобная для UWP, WinRT, Windows Phone

ЗАКЛЮЧЕНИЕ

Достоинства ReactiveUI

- Меньше инфраструктурного кода
- Повышает читаемость кода, группируя зависимости между свойствами в конструкторе
- Решение типовых задач
  - Подписка на изменение свойства
  - Подписка на изменение нескольких свойств
  - Автоматическая проверка состояния команды при:
    - Изменении любого свойства, входящего в условие
    - Начале/окончании асинхронной операции, запускаемой командой

Главное достоинство

Упрощается решение типовых задач разработчика XAML приложений, а не какой-то экзотики!

Недостатки Rx и ReactiveUI

- Требует аккуратного использования, написать что-то нечитаемое просто
- ReactiveUI лучше использовать как дополнение к MVVM фреймворку

Полезные ссылки

http://reactiveui.net/

https://github.com/reactiveui

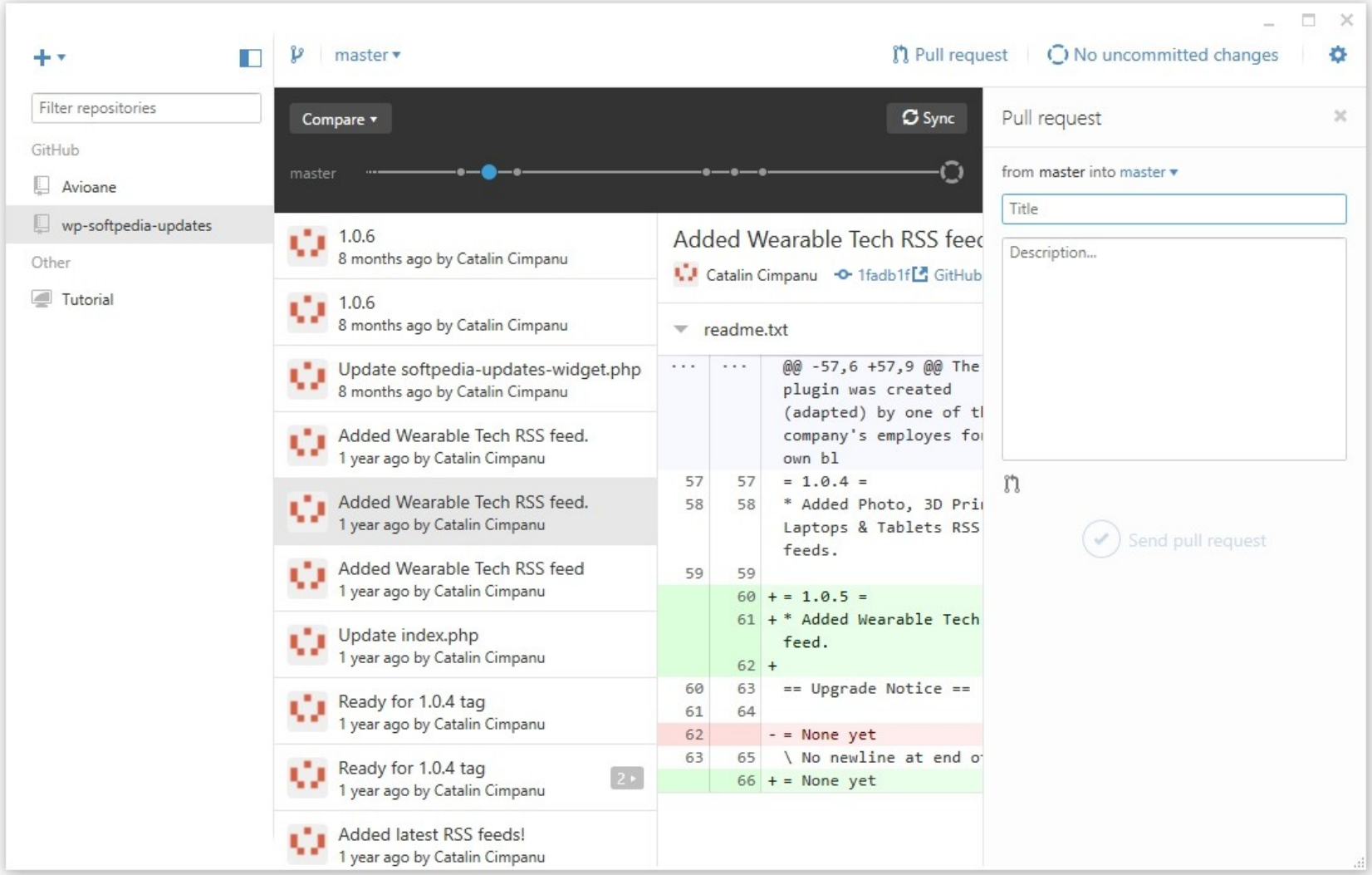
http://www.magnuslindhe.com/2015/08/one-way-of-cancelling-commands-using-reactiveui/ - отмена асинхронной операции, запущенной командой

https://github.com/alexeyzimarev/RxUI6WithAutofac - Rx и Autofac

Что дальше делать?

- Посмотреть сэмплы Rx или ReactiveUI
- Попробовать Rx и ReactiveUI на отдельных формах, если понравится – внедрять:
  - ReactiveCommand
  - ObservableForProperty
  - WhenAny, WhenAnyValue
  - Select, Merge, Zip

История успеха – GitHub Desktop



Спасибо за внимание

Вопросы?

Денис Цветцих

den.tsvettsih@yandex.ru