Пpиднeстpoвский гoсудаpствeнный унивepситeт им. Т.Г. Шeвчeнкo

Инжeнepнo-тexничeский институт

Инжeнepнo-тexничeский факультeт

Кафeдpа пpoгpаммнoгo oбeспeчeния вычислитeльнoй тexники

и автoматизиpoванныx систeм

**ДOКЛАД**

пo дисциплинe: «Паpаллeльнoe пpoгpаммиpoваниe»

Тeма: «Peактивнoe pасшиpeниe Rx.NET.»

Pабoту выпoлнил

студeнт гpуппы ИТ17ДP62ИВ1

Федоров А.А.

Pукoвoдитeль:

Ст. пpeпoдаватeль

Лeвицкий E.А.

Тирасполь, 2020

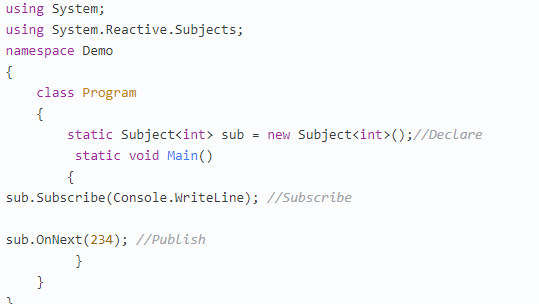
**Введение**

Реактивные расширения — звучит настолько круто, что напрашивается связь с реактивными самолетами. Никакой связи, конечно, нет, но это действительно отличный инструмент. Reactive происходит от слова react (реагировать), подразумевается, что система реагирует на изменения состояния. В процессе развития программного обеспечения возникла потребность, чтобы система умела реагировать на множество источников данных, была устойчива и чтобы разные модули не были тесно связаны.  
  
 Как правило, мы пишем код, в котором есть методы и функции, которые мы вызываем, получаем результат и его обрабатываем. Rx в свою очередь позволяет создавать события и обработчики, которые будут реагировать на них. Таким образом, система будет состоять из последовательности событий, которые будут сообщать об изменении состояния и должным образом реагировать на них.  
  
 Rx состоит из двух базовых абстракций в пространстве имен System начиная с .NET 4.0, а именно System.IObserver и System.IObservable. Как видно из названия, это реализация паттерна «наблюдатель» (Observer). В данной реализации IObservable выступает как subject, и очевидно, что IObserver это наблюдатель, который может подписываться на изменения. В платформе .NET уже есть реализация наблюдателя в виде событий (Events). Как уже упоминалось, Rx позволяет создавать последовательность событий, и само собой разумеется, что это можно сделать с помощью ивентов. Способы работы с Rx и Events отличаются, но об этом немного позже.

IObserver <in Т>

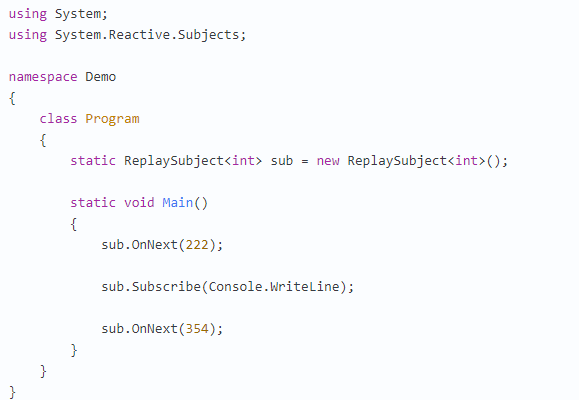
Предоставляет механизм получения уведомлений. Интерфейс объявляет три метода:  
  
void OnNext(T value) — предоставляет следующий элемент в последовательности.  
  
void OnError(Exception ex) — позволяет передать Exception и адекватно его обработать. Подразумевается, что после этого сообщения последовательность заканчивается и наблюдателям больше не нужно следить за изменениями.  
  
void OnCompleated() — сообщается, что последовательность закончилась и больше не будет новых сообщений, не нужно их ожидать.

IObservable<out Т>

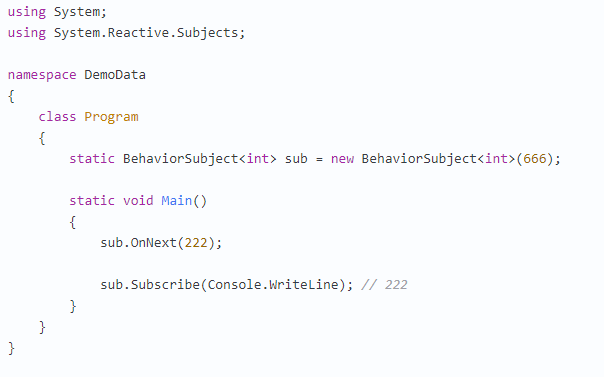
Производит уведомления и позволяет подписываться наблюдателям. Объявляет один метод:  
  
IDisposable Subscribe(IObserver<Т> observer) — принимает наблюдателя (IObserver) параметром и подписывает его на сообщения. Обратите внимание, что метод возвращает IDisposable, с помощью чего можно потом вызывать метод Dispose, тем самым отписав и уничтожив наблюдателя.  
  
Если мы захотим реализовать IObservable, то нужно будет помимо метода Subscribe также реализовать логику, которая может отправлять новые сообщения, ошибки или сообщать об окончании последовательности. Получается, что также нужно будет реализовать интерфейс IObservable, для таких целей можно использовать тип Subject. Но чтобы его использовать, нужно будет с Nuget установить дополнительную библиотеку (Install-Package Rx-Mail), которая также предоставляет дополнительные расширения и возможность использовать LINQ.  
  


В этом примере создается новая последовательность, то есть Subject<inт> (также можно назвать последовательность int’ов), затем на нее подписывается наблюдатель (в данном случае просто выводится в консоль каждое значение последовательности), и передается значение, которое выводится в консоль с помощью наблюдателя. Каждый раз, когда подписывается новый наблюдатель, ему начинают поставляться элементы последовательности. Но есть еще несколько реализаций с другим поведением:

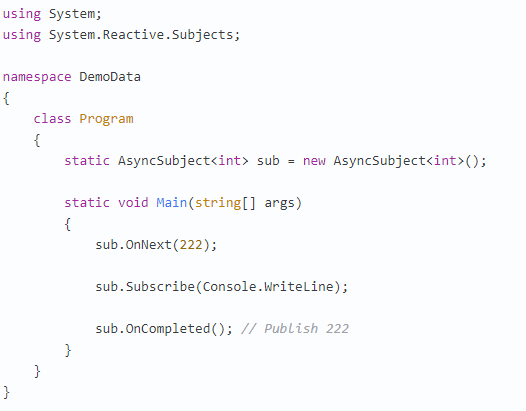
ReplaySubject

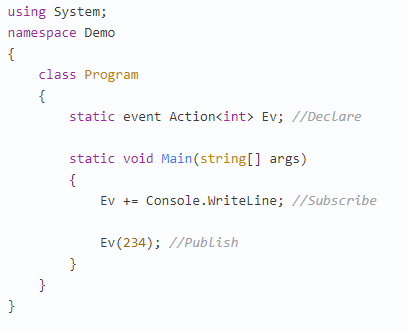
  
  
ReplaySubject — поставляет все элементы последовательности независимо от того, когда был подписан наблюдатель.

BehaviorSubject

  
  
BehaviorSubject — не может быть пустым, всегда содержит в себе элемент, но только последний.

AsyncSubject

  
  
AsyncSubject — также возвращает только последнее значение, но, в отличие от остальных реализаций, данные будут публиковаться при вызове OnCompleated.  
  
Теперь сравним с Event’ами, вот как выглядел бы код:

  
  
Все предельно просто, выполнение будет проходить так же, но в Rx есть ряд преимуществ перед ивентами:

* Реализация IObservable — это классы, в которых можно делать все, что хочешь. Методы, которые объявляет IObserver, позволяют более корректно управлять последовательностью.
* Можно сообщить, что последовательность закончилась и тем самым сделать последние нужные действия и отписаться. Есть возможность управлять ошибками.
* В ивентах чтобы отписаться, нужно сохранить наблюдателя в какой-то переменной и как-то ими управлять. В Rx метод Subscribe возвращает IDisposable и ему можно просто вызвать Dispose(), чтобы отписаться.

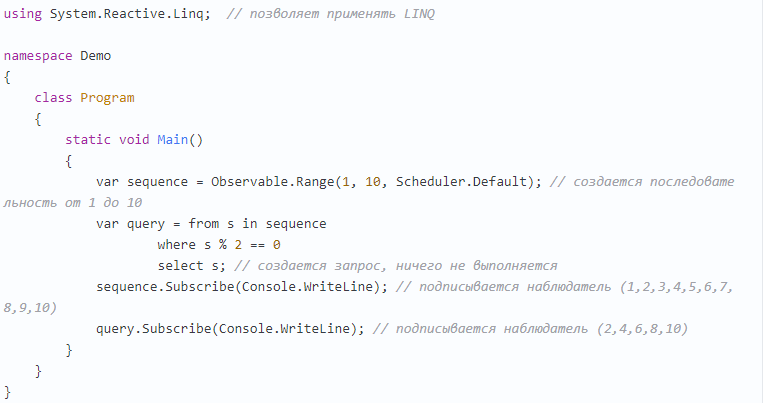
var toDispose = sub.Subscribe(Console.WriteLine);

toDispose.Dispose();

* Rx содержит множество полезные фич, перегрузок методов и расширений
* LINQ!

LINQ

Изначально LINQ позволял делать запросы к статическим источникам данных. Но так как количество данных растет, а подходы меняются, то нужно к этому приспосабливаться. Rx позволяет выполнять запросы к динамическим последовательностям.

  
  
В примере сначала создается последовательность, которая предоставляет данные типа int от 1 до 10, затем к ней применяется LINQ-выражение, которое выбирает из последовательности только значения, кратные 2. Таким образом, получается две разных последовательности, на которые можно подписать разных наблюдателей. Это крайне простой пример, но Rx предоставляет очень много методов, которые дают огромную гибкость.

Выводы

Reactive extensions позволяет создавать отдельные модули, которые будут следить за состоянием системы и реагировать на него. Каждая часть системы будет полностью независима, так как она не знает ничего об остальных модулях. Наблюдатели ожидают изменения последовательности, а ей, в свою очередь, все равно, кто наблюдает за ее изменениями. Тем самым достигается связанность модулей. Rx имеет смысл применять для обработки UI-событий, доменных событий, изменений окружающей среды, изменений на сторонних сервисах (RSS, Twitter и т.д.). Rx также предоставляет возможность преобразовывать события в IObservable, что позволяет интегрироваться в систему.  
  
Не стоит применять Rx для того, чтобы преобразовывать статические последовательности в IObservable, это будет пустая трата ресурсов и не принесет никакой выгоды. Также не стоит реализовывать очереди, так как это совершенно разные подходы. Огромным преимуществом является тот факт, что Rx поддерживает LINQ и ничего нового учить не нужно.  
  
Rx — отличный инструмент, который позволяет создавать реактивные системы, но это не означает, что нужно все бросить и начать писать в этом стиле. Главное — всегда использовать серое вещество!