为什么使用Rx库：从基于拉取的方式变更为基于推送的方式，可观察集合的实现包含在Rx库中。

将普通集合转化为可观察集合：

文本

描述已自动生成

图形用户界面, 文本

描述已自动生成

使用ToObservable方法把一个效率不高的可枚举集合转化为可观察集合，然后订阅该集合，把Console.Write方法作为操作，在每次更新该集合时执行。为了异步化该程序，然后使用SubscribeOn方法并传入TPL任务池调度程序，该调度把订阅信息放入任务池中并卸除主线程的任务，因此可以在集合更新时保持相应并做其他事。

Ps：

1.SubscribeOn方法是用于指定在哪个调度器上执行订阅操作的方法。

2. SubscribeOn方法可以用于以下情况：

异步操作：如果订阅操作涉及到耗时的异步任务，使用SubscribeOn可以将其放在后台线程或任务池中执行，避免阻塞主线程。

线程切换：如果需要在特定的线程或调度器上执行订阅操作，可以使用SubscribeOn指定相应的调度器，例如在IO线程或计算线程上执行。

并行执行：如果有多个订阅操作需要同时执行，可以使用SubscribeOn将它们分配到不同的线程或任务池中，实现并行执行的效果。

编写自定义的可观察对象：

需要实现IObserver<in T>和IObservable<out T>接口。IObservable<out T> 适用于消费数据的场景，泛型参数 T 是逆变的，可以接受 T 的基类作为参数。IObserver<in T>适用于生产数据的场景，泛型参数 T 是协变的，可以返回 T 或 T 的派生类。

IObserver<in T>

文本

中度可信度描述已自动生成

图形用户界面, 文本, 应用程序

描述已自动生成

IObservable<out T>:

图形用户界面, 文本, 应用程序

描述已自动生成

图形用户界面, 文本, 应用程序

描述已自动生成

下面是书上的例子：

文本, 信件

描述已自动生成

文本

描述已自动生成

图形用户界面, 文本, 应用程序

描述已自动生成

图形用户界面, 文本, 应用程序

描述已自动生成

这是执行结果：

图形用户界面, 文本, 应用程序

描述已自动生成

例子中创建的实例badObservable传入空的集合，但是并没有执行执行OnError方法进行通知错误的发生。OnError需要手动调用，书上的解释是有误的，需要添加这段代码才会触发OnError通知：

文本

描述已自动生成

使用Subject类型：

Subject类是一个特殊的可观察对象（Observable）和观察者（Observer）的组合，它充当了一个桥梁或代理的角色。Subject既可以作为数据源（Observable），也可以作为数据的订阅者（Observer）。

Subject类的具体作用：

Subject作为可观察对象（Observable）：Subject可以订阅一个或多个可观察对象，并将其观测到的数据对象传递给订阅它的观察者。当Subject接收到新的数据时，它会遍历观察者列表，并依次调用它们的OnNext方法，将数据传递给它们。

Subject作为观察者（Observer）：Subject可以订阅一个或多个可观察对象，并将自身作为观察者添加到观察者列表中。当Subject订阅的可观察对象发送新的数据时，Subject会接收到这些数据，并通过调用自身的OnNext方法将数据传递给自身的观察者。

Ps：当我们使用Subject或其他可观察对象作为数据源时，我们需要在适当的时机调用OnCompleted方法来表示数据流的结束。通常，在我们确定不会再有新的数据产生时，或者达到了预定的结束条件时，我们可以调用OnCompleted方法。

Subject类有多个具体的实现：

BehaviorSubject：在订阅时会发送最近的数据，并在没有新数据时重复发送最后一个数据。

ReplaySubject：在订阅时会发送所有的历史数据，可以设置缓存的大小。

AsyncSubject：在完成时发送最后一个数据，如果没有数据则不发送。

这是其中比较特殊的AsyncSubject的用法，它在关闭数据流时才返回最后一个订阅时（最后一个OnNext）执行的情况

文本

描述已自动生成

Ps：从下面这个例子可以得知Subject和他的三个具体实现类之间的订阅互不干扰：

文本

描述已自动生成

上面这个例子有一个问题，我忘记调用OnCompleted了，忘记做这件事会带来危害，如果在使用Rx库中的Subject或Observable时忘记调用OnCompleted方法，观察者将无法收到数据流的结束通知。这可能会导致观察者一直等待数据流的结束，从而造成潜在的资源泄漏或阻塞。

创建可观察对象：

创建可观察对象可以使用Observable类的静态方法，比如Observable.Create、Observable.FromEvent和Observable.Interval还有Observable.Return、Observable.Empty<T>、Observable.Throw<T>、Observable.Repeat、Observable.Range等

Create:

文本

描述已自动生成

Return:

文本

描述已自动生成

Empty

文本

描述已自动生成

Throw

文本

描述已自动生成

Repeat:

文本

描述已自动生成

Range:

文本

描述已自动生成

Generate:

图形用户界面, 文本

描述已自动生成

FromEvent:

文本

描述已自动生成

Interval:

文本

描述已自动生成

对可观察集合使用Linq查询：

下面是书上的例子和解释：

文本, 信件

描述已自动生成

文本

描述已自动生成

文本

描述已自动生成

上面书上的简明的例子展示了Linq可以对可观察集合并行查询，通过并行的方式查询并以此来提高查询效率

使用Rx创建异步操作：

下面是书上例子各个部分用到的方法：

手机屏幕截图

描述已自动生成

文本

描述已自动生成

下面是例子的第一部分：

图形用户界面, 文本, 电子邮件

描述已自动生成

这个片段中有一个Observable.Start方法，效果与Task.Run方法相似。Observable.Start方法接受一个委托作为参数，该委托表示要在后台执行的操作。它会将操作封装在一个Task中，并返回一个表示该任务的可观察对象。

下面是例子的第二部分：

文本, 电子邮件

描述已自动生成

这个片段中有一个ToObservable扩展方法，把任务转化为Observable方法。，可以将其他类型的对象或数据源转换为可观察对象（例如Task，事件，集合类型）。它提供了一种将非可观察对象转换为可观察对象的便捷方式

下面是例子的第三部分：

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

使用Observable.FromAsyncPattern方法将APM直接转化为可观察对象

下面是例子的第四部分：

文本

描述已自动生成

可以对可观察对象使用await操作符

下面是例子的第五部分：

图形用户界面, 文本, 应用程序

描述已自动生成

把基于事件的异步模式直接转化为可观察对象