Programación funcional, un nuevo enfoque

Alex Casquete
Email: acasquete@outlook.com
LinkedIn: https://www.linkedin.com/in/alexcasquete/

Febrero 2015

Resumen

Aunque la programación funcional no es un concepto nuevo ni reciente, en los últimos años ha comenzado a tener más protagonismo dentro de la comunidad de desarrolladores. En este cambio de tendencia han influido indudablemente los nuevos desafíos a los que los desarrollos actuales tienen que hacer frente. En la era del Big Data, las aplicaciones modernas deben de ser capaces de procesar grandes cantidades de información, poder ejecutar procesos de forma concurrente, escalar a múltiples procesadores y, además, necesitamos que todo esto pueda ser probado fácilmente. A todas estas necesidades, los lenguajes funcionales aportan una nueva respuesta con un nuevo enfoque.

Nota: Este artículo se publicó el 12 de febrero de 2015 en MSDN España (https://web.archive.org/web/20150418061308/http://blogs.msdn.com/b/esmsdn/archive/2015/02/13/post-invitado-programaci-243-n-funcional-un-nuevo-enfoque.aspx)

Abstract

Although functional programming is not a new or recent concept, it has begun to have more prominence within the developer community in recent years. This change in trend has undoubtedly been influenced by the new challenges that current developments must face. In the era of Big Data, modern applications must be able to process large amounts of information, run processes concurrently, scale to multiple processors, and we also need all of this to be easily proven. To all these needs, functional languages provide a new answer with a new approach.

Note: This article was published on February 12, 2015 on MSDN Spain (https://web.archive.org/web/20150418061308/http://blogs.msdn.com/b/esmsdn/archive/2015/02/13/post-invitado-programaci-243-n-funcional-un-nuevo-enfoque.aspx)

Introducción

Pensemos, por ejemplo, en cómo podría funcionar un sistema de *ticketing* que procesa una lista de billetes a imprimir. En primer lugar obtendríamos un listado de los billetes para poder iterar por cada uno ellos. Después comprobaríamos si cada uno cumple las condiciones para ser impreso, en caso afirmativo lo enviaríamos a imprimir y continuaríamos con la siguiente iteración. La programación funcional nos ayuda a plantear el problema de una forma distinta. Si pensamos de una forma declarativa el planteamiento podría ser "enviar a imprimir todos los tickets que se pueden imprimir". En la primera forma, utilizando la programación imperativa, nuestro código dice mediante una secuencia de comandos exactamente cómo se tiene que hacer una operación. Por el contrario, mediante la programación funcional tenemos una expresión que nos dice qué es lo que se tiene que hacer. Con este sencillo ejemplo, podemos ver el cambio de enfoque al que nos vemos obligados cuando utilizamos otro paradigma de programación. Pero además, la diferencia entre los dos modelos de programación será mucho más marcada a medida que la complejidad de nuestra aplicación crezca.

Programación imperativa	Programación declarativa

foreach (Ticket t in Tickets)
{
  if (t.CanBePrinted())
  {
  t.Print();
  }
}
	Tickets
	  .Where(t => t.CanBePrinted())
	  .ForEach(t => t.Print());

Listado 1 Programación imperativa comparada con la programación declarativa

Como podemos ver en el Listado 1, muchos lenguajes ya han incorporado algunas características propias de los lenguajes funcionales. Este cambio no solo se ha producido en el ecosistema tecnológico de Microsoft, pero si ponemos como ejemplo el desarrollo con .NET, nos encontramos con que los genéricos, los métodos anónimos, las expresiones lambda o LINQ son claros ejemplos de características influenciadas por la programación funcional que ya forman parte de C# o Visual Basic. No obstante, a pesar de estas características funcionales, estos lenguajes siguen siendo orientados a objetos.

F#, un lenguaje *functional-first*

En 2002, Don Syme y su equipo en *Microsoft Research* en Cambrige diseñaron F#, un lenguaje inspirado por OCaml, que es su vez de la familia de los lenguajes ML y posee algunas características de Haskell. La versión 1.0 de F# no apareció hasta 2005 y la versión 2.0 fue la primera versión estabilizada con soporte completo en Visual Studio 2010. En 2012, se liberó la versión 3.0 con Visual Studio 2012 y la 3.1 con Visual Studio 2013. La última versión disponible del lenguaje es una *preview* de la versión 4.0 que se ha distribuido con la también preview de Visual Studio 2015.

F# es el lenguaje de la plataforma .NET verdaderamente funcional y, aunque está soportado oficialmente desde 2010, podemos decir que nos encontramos ante un lenguaje maduro. A diferencia de otros lenguajes funcionales, F# es un lenguaje multiparadigma *functional-first*. Esto significa que facilita que la programación funcional sea la primera opción utilizada para resolver la mayoría de los problemas de programación, pero también admite la programación imperativa y la orientación a objetos con soporte completo del modelo de objetos de .NET.

Hoy en día, F# está siendo utilizado en una amplia gama de áreas de aplicación como modelos financieros, análisis estadístico y *machine learning*, donde ha demostrado mejoras de productividad con respecto a la programación "tradicional". Pero aparte de destacar en aplicaciones ricas en datos, F# ha ganado terreno muy rápidamente en otros ámbitos, sobre todo gracias al soporte de la comunidad y de múltiples empresas líderes en la industria. Un ejemplo de esta diversificación es el soporte de Xamarin al lenguaje, permitiendo crear aplicaciones móviles nativas multiplataforma utilizando F# como lenguaje principal.

Sin embargo, la adopción en la empresa está siendo lenta, principalmente debido a la falta de profesionales con sólidos conocimientos de programación funcional, lo que lleva a las empresas a descartar el uso de un nuevo paradigma de programación para reducir el riesgo que podría suponer el incluirlo en sus ciclos de desarrollo.

Una nueva forma de programar

La dificultad principal cuando nos enfrentamos con un paradigma de programación totalmente nuevo no es el lenguaje, sino el aprender a pensar de una forma diferente. Es importante entender que la programación funcional no es simplemente un cambio estético con cambios en la sintaxis, conlleva una nueva forma de pensar en los problemas.

En el caso de F#, al tratarse de un lenguaje multiparadigma, tenemos la ventaja de poder elegir que enfoque aporta la mejor solución a un problema determinado, pero también implica que al poder utilizar también la programación imperativa, se hace muy tentador el pensar siempre de forma imperativa. Si queremos aprovechar los beneficios de la programación funcional, es necesario que realicemos un cambio en la forma de plantear los problemas.

Si pensamos, por ejemplo, en algunas construcciones de la programación orientada a objetos, como la encapsulación o la visibilidad, nos daremos cuenta de que todo existe para tener un control sobre quien puede ver y cambiar el estado. En lugar de tener estos mecanismos para controlar los estados mutables, los lenguajes funcionales intentan eliminarlos. La idea detrás de esta decisión es que si el lenguaje expone menos características propensas a errores, los desarrolladores cometeremos menos errores.

¿Qué es un lenguaje funcional?

Aunque muchas veces existen distintos criterios para establecer qué características debe tener un lenguaje para considerarlo funcional, existen una serie de conceptos que todos los lenguajes funcionales comparten, estos son: la inmutabilidad, *functions higher-order* o funciones de orden superior y la recursividad.

Inmutabilidad

Cuando pensamos en variables normalmente pensamos en variables mutables, después de todo, una variable es eso, variable. La inmutabilidad es una característica del lenguaje que establece que un objeto no puede ser cambiado una vez creado. En F# muchos de los valores y estructuras de datos son inmutables, es decir que ni el valor local ni su contenido se pueden cambiar. En C# es posible replicar la mismas construcciones inmutables, pero la diferencia está en que en F# es inmutable por defecto.

Inmutabilidad en C#	Inmutabilidad en F#

class Point
{
  private readonly int x, y;
  public int X { get { return x; } }
  public int Y { get { return y; } }
  public Point(int x, int y)
  {
  this.x = x;
  this.y = y;
  }
}
 	
var p = new Point(1, 2);
	type Point = { x: int; y: int }
	
	let p = { x = 1; y = 2 }

Listado 2 Comparación de la definición de un tipo inmutable en C# y F#

Aunque parece que con la inmutabilidad es difícil construir aplicaciones complejas, dado que el estado de una aplicación debe cambiar en algún momento, la inmutabilidad nos ofrece dos ventajas principales. La primera es que si pasamos un objeto a una función, tendremos la certeza de que ese objeto no cambiará de estado, con lo que conseguimos una mejora en la lectura del código ya que para saber lo que hace una función solo tendremos que analizar los parámetros de entrada y el valor devuelto. La segunda ventaja es que podemos acceder a objetos inmutables de forma concurrente sin preocuparnos de que se puedan corromper. Ningún hilo puede provocar efectos secundarios en un objeto inmutable, por lo tanto, un código que utiliza objetos inmutables es mucho más sencillo de paralelizar.

Funciones de orden superior

Las funciones de orden superior son funciones que pueden aceptar otras funciones como argumentos o devolver una función. En la programación orientada a objetos pensamos en los problemas en términos de objetos, definimos clases. En la programación funcional, las funciones son las construcciones principales, podríamos decir que las funciones se tratan como objetos. El uso de funciones, devolverlas o pasarlas a otras funciones se convierte en algo extremadamente útil en cuanto a la reusabilidad y abstracciones del código, son muy útiles para refactorizar y reducir el código. Por ejemplo, la mayoría de bucles **for** se pueden expresar utilizando funciones de mapeo. Un ejemplo básico de una función de orden superior es **map** (Listado 3), que espera como argumentos una función y una lista, y aplica la función a todos los elementos de lista.

```
List.map (fun x -> x * 2) [0..100]
devuelve
[0, 2, 4, 6, 8, 10, 12, ...]
```

Listado 3 Ejemplo de uso y resultado de la función map de F#

Recursividad

En la programación imperativa, cuando queremos iterar, utilizamos construcciones de tipo **for** o **while**. Debido a que este tipo de construcción se basa en cambios de estado para saber en

qué momento hay que salir del bucle, en la programación funcional tenemos que hacer uso de otro enfoque.

Versión iterativa en C#	Versión recursiva en F#

int fib(int n)
{
  int fib0 = 0, fib1 = 1;
  for (int i = 2; i <= n; i++)
  {
  int aux = fib0;
  fib0 = fib1;
  fib1 = aux + fib1;
  }
  return (n > 0 ? fib1 : 0);
}
	let rec fib n =
	  if n < 2 then n
	  else fib(n - 1) + fib(n - 2)

Listado 4 Cálculo del número Fibonacci en forma iterativa en C# comparada con la versión recursiva en F#

En F# y en la programación funcional en general utilizamos funciones recursivas en las que una función se llama a si misma directa o indirectamente. Al ser un mecanismo esencial de la programación funcional, los lenguajes funcionales tienen varios mecanismos de optimización para evitar desbordamientos de pila al realizar múltiples llamadas recursivas.

Hecho por y para la comunidad

Desde que se abrieron las contribuciones de la comunidad en abril del pasado año, gran parte del nuevo contenido, tanto del lenguaje F# como del conjunto de herramientas para el IDE, ha sido desarrollado por contribuyentes que no pertenecen a Microsoft, en plena colaboración con la comunidad de desarrolladores *open-source*. En tan sólo este corto tiempo, el proyecto de Visual F# ha recibido multitud de *pull requests* de varias decenas de desarrolladores, de las que se han podido aplicar una gran parte de ellas en la última versión.

En España, hasta hace pocos meses, la comunidad de desarrolladores de F# era casi testimonial, muy al contrario que en otros países de nuestro entorno como Francia o el Reino Unido, donde se vienen organizando eventos dedicados a F# con regularidad desde hace años. Sin embargo, parece que está situación está cambiando. Actualmente existen varios grupos de usuarios en Barcelona y en Madrid con una actividad regular y cada vez son más los desarrolladores que muestran interés en conocer los beneficios de la programación funcional y de F# en particular.

Estos dos grupos se coordinan a través de sus respectivas páginas en Meetup. En el caso del Barcelona F# Meetup (http://bit.ly/fsbcn), inició su actividad oficial el pasado diciembre y ha organizado varios eventos dedicados a mostrar los conceptos fundamentales de la programación funcional con F#, las estructuras de datos básicas y los proveedores de tipo. Este grupo tiene previsto realizar reuniones mensuales en la que se realizarán distintas actividades: presentaciones técnicas, workshops prácticos y coding dojos.

El Madrid F# Meetup (http://bit.ly/fsmadrid) nació al mismo tiempo que el grupo de Barcelona y, gracias al apoyo de Kaleidos y el meetup Functional Programming Madrid, ha celebrado un par de reuniones, una sesión introductoria a F# y otra sobre el proyecto FunScript, un compilador de F# a JavaScript. Durante este año continuará organizando charlas y estrechando la colaboración con otras

comunidades tecnológicas de programación funcional o .NET en general. El plato fuerte lo servirá Riccardo Terrel, el organizador del grupo de F# en Washington que les hará una visita el próximo 31 de marzo para hablar de F# y la web.

Tanto si vives cerca de Madrid o de Barcelona, estos dos grupos son una fantástica oportunidad para entrar en contacto con una creciente comunidad de usuarios interesados en F# y en la programación funcional. Además puede ser la forma de colaborar en algún proyecto de código abierto y contribuir con un pequeño grano de arena a la gran comunidad global de F#.

Siguientes pasos para aprender F#

Dejando de lado que el estilo de la programación funcional sea más elegante, el motivo principal por el que deberíamos empezar a considerar F# como lenguaje para nuestro próximo desarrollo debería ser exclusivamente por motivos de productividad. La ausencia de efectos colaterales, mayor facilidad para la ejecución concurrente y pruebas unitarias más confiables son solo algunas de las ventajas que nos aporta la programación funcional.

Para iniciarse en la programación funcional y en F# en particular, tenemos a nuestra disposición multitud de material de estudio, tanto libros y artículos técnicos, como videos, blogs y cursos online. La forma más rápida para introducirnos en el lenguaje es mediante "Try F#" (http://www.tryfsharp.org), una herramienta online que, además de contener muchísima información sobre el lenguaje, nos ofrece muestras de código que podemos modificar y ejecutar directamente en el navegador. Otro recurso online imprescindible es "F# for fun and profit" (http://fsharpforfunandprofit.com/), donde encontraremos tutoriales y documentación, tanto de conceptos básicos como de otros más avanzados. Por último, en la *Curah!* "Programación funcional con F#" (http://bit.ly/curahfsharp) encontraremos una recopilación de artículos escritos en español sobre programación funcional con F#. Todos ellos, recursos imprescindibles para el que quiera conocer y profundizar en la programación funcional con F#.

Bibliografía

Backfield, Joshua. *Becoming functional*. Sebastopol: O'Reilly Media, 2014. ISBN 978-1-449-36817-3

Ford, Neal. *Functional Thinking*. Sebastopol: O'Reilly Media, 2014. ISBN 978-1-449-36551-6

Syme, Don; Granicz, Adam; Cisternino, Antonio. *Expert F# 3.0*. 3a edición. New York: Apress, 2012. ISBN 978-1-4302-4650-3

Petricek, Tomas; Skeet, Jon. *Real-World Functional Programming*. New York: Manning Publications, 2009. ISBN 978-1933988924