PureScript mediante ejemplos

Programación funcional para la Web

Por Phil Freeman
Traducido por Jorge Acereda

PureScript mediante ejemplos

Programación funcional para la Web

Phil Freeman y Jorge Acereda

Este libro está a la venta en http://leanpub.com/purescriptmedianteejemplos

Esta versión se publicó en 2017-09-28

Este es un libro de Leanpub. Leanpub anima a los autores y publicadoras con el proceso de publicación. Lean Publishing es el acto de publicar un libro en progreso usando herramientas sencillas y muchas iteraciones para obtener feedback del lector hasta conseguir tener el libro adecuado.

Índice general

Introducción

JavaScript funcional

Hace ya algún tiempo que han empezado a aparecer las técnicas de programación funcional en Javascript:

- Bibliotecas como UnderscoreJS[1] permiten al desarrollador aprovechar funciones probadas como `map`, `filter` y `reduce` para crear programas más grandes a partir de programas más pequeños mediante composición:

```
var sumOfPrimes =
    _.chain(_.range(1000))
     .filter(isPrime)
     .reduce(function(x, y) {
         return x + y;
     })
     .value();
```

- La programación asíncrona en NodeJS se apoya firmemente en las funciones como valores de primera clase para definir retrollamadas (*callbacks*).

```
require('fs').readFile(sourceFile, function (error, data) {
  if (!error) {
    require('fs').writeFile(destFile, data, function (error) {
      if (!error) {
        console.log("File copied");
      }
    });
  }
});
```

- Bibliotecas como React[2] y virtual-dom[3] modelan las vistas como funciones puras sobre el estado de la aplicación.

[1] http://underscorejs.org
[2] http://facebook.github.io/react/
[3] https://github.com/Matt-Esch/virtual-dom

Las funciones permiten una forma simple de abstracción que puede resultar en grandes ganancias en productividad. Sin embargo, la programación funcional en JavaScript tiene sus propias desventajas. JavaScript es verboso, no tipado, y carece de formas potentes de abstracción. El JavaScript no restringido hace también el razonamiento ecuacional muy difícil.

PureScript es un lenguaje de programación cuyo objetivo es abarcar estos problemas. Proporciona sintaxis ligera, que nos permite escribir código muy expresivo que sigue siendo claro y legible. Usa un rico sistema de tipos para soportar abstracciones potentes. También genera código rápido e inteligible, cosa importante cuando hay que interoperar con JavaScript u otros lenguajes que compilan a JavaScript. Total, espero convencerte de que PureScript consigue un equilibrio muy práctico entre el poder teórico de la programación funcional pura y el estilo de programación rápida y flexible de JavaScript.

Tipos e inferencia de tipos

El debate sobre los lenguajes de tipado estático contra los de tipado dinámico está bien documentado. PuseScript es un lenguaje de *tipado estático*, lo que significa que el compilador puede dar a un programa correcto un *tipo* que indica su comportamiento. A la inversa, los programas a los que no se les puede dar un tipo son *programas incorrectos*, y serán rechazados por el compilador. En PureScript, al contrario que en los lenguajes de tipado dinámico, los tipos existen únicamente en *tiempo de compilación*, y no tienen representación en tiempo de ejecución.

Es importante notar que, de varias maneras, los tipos en Purescript no son como los tipos que has visto en otros lenguajes como Java o C#. Aunque sirven el mismo propósito a un nivel alto, los tipos en PureScript están inspirados por lenguajes como ML y Haskell. Los tipos de PureScript son expresivos, permitiendo al desarrollador hacer afirmaciones sólidas sobre sus programas. Más importante, el sistema de tipos de PureScript soporta *inferencia de tipos* requiriendo muchas menos anotaciones de tipo explícitas que otros lenguajes, convirtiendo el sistema de tipos en una *herramienta* en lugar de en un estorbo. Un simple ejemplo, el siguiente código define un *número*, pero no hay mención del tipo `Number` en ningún sitio del código:

```
1  iAmANumber =
2    let square x = x * x
3    in square 42.0
```

Un ejemplo más elaborado muestra que la corrección de los tipos puede ser confirmada sin anotación de tipos, incluso cuando existen tipos que son *desconocidos para el compilador*:

```
1  iterate f 0 x = x
2  iterate f n x = iterate f (n - 1) (f x)
```

Aquí, el tipo de `x` es desconocido, pero el compilador sigue pudiendo verificar que `iterate` obedece las reglas del sistema de tipos, sin importar qué tipo pueda tener `x`.

En este libro, intentaré convencerte (o reafirmar tu creencia) de que los tipos estáticos no sólo son medios para ganar confianza en la corrección de tus programas, sino que también ayudan al desarrollo por derecho propio. Refactorizar una base de código extensa en JavaScript puede ser difícil para cualquier cosa que no sean las abstracciones más simples, pero un sistema de tipos expresivo junto a un comprobador de tipos puede incluso convertir la refactorización en una experiencia divertida e interactiva.

Además, la red de seguridad proporcionada por un sistema de tipos permite formas de abstracción más avanzadas. De hecho, PureScript proporciona una poderosa forma de abstracción que es fundamentalmente guiada por tipos: las clases de tipos (*type classes*) populares en el lenguaje de programación funcional Haskell.

Programación web políglota

La programación funcional tiene sus historias de éxito, aplicaciones donde ha sido particularmente exitosa: análisis de datos, análisis sintáctico, implementación de compiladores, programación genérica o paralelismo por nombrar unas cuantas.

Sería posible practicar desarrollo de aplicaciones de extremo a extremo en un lenguaje funcional como PureScript. PureScript proporciona la habilidad de importar código JavaScript existente, proporcionando tipos para sus valores y funciones, y usar entonces esas funciones en código PureScript normal. Veremos este enfoque más tarde en el libro.

Sin embargo, una de las fortalezas de PureScript es su interoperabilidad con otros lenguajes que compilan a JavaScript. Otro enfoque sería usar PureScript como un subconjunto del desarrollo de tu aplicación y usar otro(s) lenguajes para escribir el resto del código JavaScript.

Aquí hay algunos ejemplos:

- Lógica central escrita en PureScript, con la interfaz de usuario escrita en JavaScript.
- Aplicación escrita en JavaScript u otro lenguaje que compile a JavaScript, con pruebas escritas en PureScript.
- PureScript usado para automatizar las pruebas de interfaz de usuario para una aplicación ya existente.

En este libro, nos vamos a enfocar en resolver pequeños problemas con PureScript. Las soluciones se podrían integrar en una aplicación más grande, pero también veremos como llamar a código PureScript desde JavaScript y viceversa.

Prerrequisitos

Los requerimientos de software para este libro son mínimos: El primer capítulo te guiará para preparar un entorno de desarrollo desde cero, y las herramientas que vamos a usar están disponibles en los repositorios estándar de la mayoría de sistemas operativos modernos.

El compilador de PureScript se puede descargar como una distribución binaria o se puede construir a partir de los fuentes en cualquier sistema que tenga una instalación reciente del compilador de Haskell GHC, el siguiente capítulo dará los pasos.

El código de esta versión es compatible con las versiones `0.11.*` del compilador de PureScript.

Sobre ti

Voy a asumir que estas familiarizado con JavaScript. Cualquier familiaridad previa con herramientas comunes del ecosistema JavaScript como NPM y Gulp serán beneficiosas si deseas adaptar la configuración estándar para tus necesidades, pero dicho conocimiento no es necesario.

No se necesita ningún conocimiento previo de programación funcional, pero ciertamente no hace daño. Las ideas nuevas estarán acompañadas de ejemplos prácticos, de manera que seas capaz de formar una intuición para los conceptos de programación funcional que usaremos.

Los lectores que estén familiarizados con el lenguaje de programación Haskell reconocerán un montón de las ideas y sintaxis presentadas en este libro, ya que PureScript está fuertemente influenciado por Haskell. Sin embargo, dichos lectores deben entender que hay unas cuantas diferencias importantes entre PureScript y Haskell. No siempre va a ser apropiado intentar aplicar ideas de un lenguaje en el otro, aunque muchos de los conceptos presentados aquí tendrán una interpretación en Haskell.

Cómo leer este libro

Los capítulos de este libro son bastante autocontenidos. Sin embargo, un principiante con poca experiencia en programación funcional debería estudiar los capítulos en orden. Los primeros capítulos sientan las bases requeridas para entender el material que aparece más tarde en el libro. Un lector que se sienta cómodo con las ideas de la programación funcional (especialmente uno con experiencia en un lenguaje fuértemente tipado como ML o Haskell) probablemente será capaz de adquirir una comprensión general del código en los capítulos posteriores del libro sin leer los capítulos iniciales.

Cada capítulo se va a enfocar en un único ejemplo práctico, proporcionando la motivación para cualquier idea nueva introducida. El código de cada capítulo está disponible en este repositorio GitHub[4]. Algunos capítulos incluirán fragmentos de código tomados del código fuente del capítulo,

[4] https://github.com/paf31/purescript-book

pero para adquirir un entendimiento completo debes leer el código del repositorio junto al material del libro. Las secciones más largas contendrán fragmentos más cortos que puedes ejecutar en el modo interactivo (PSCi) para ayudarte a entender.

Los ejemplos de código aparecerán en una fuente monoespaciada, como sigue:

```
module Example where

import Control.Monad.Eff.Console (log)

main = log "Hello, World!"
```

Los comandos que deben escribirse en la línea de comandos estarán precedidos por un símbolo de dólar:

```
$ pulp build
```

Normalmente, estos comandos estarán orientados a usuarios de Unix, de manera que los usuarios de Windows tendrán que hacer pequeños cambios como modificar el separador de ficheros o reemplazar los comandos empotrados de shell con sus equivalentes de Windows.

Los comandos que se deben escribir en el indicador de comandos de modo interactivo de PSCi estarán precedidos por un signo ‘mayor que’.

```
> 1 + 2
3
```

Cada capítulo contendrá ejercicios etiquetados con su nivel de dificultad. Es muy recomendable que intentes hacer los ejercicios de cada capítulo para entender el material completamente.

Este libro pretende proporcionar una introducción al lenguaje PureScript para principiantes, pero no es la clase de libro que proporciona una lista de soluciones ‘plantilla’ para problemas. Para los principiantes, este libro debe ser un reto divertido, y lograrás sacarle el mayor partido si lees el material, intentas hacer los ejercicios y, lo más importante, intentas escribir algún código por tu cuenta.

Consiguiendo ayuda

Si te atascas en algún punto, hay un número de recursos disponibles online para aprender PureScript:

- El canal IRC de PureScript es un sitio estupendo para hablar sobre los problemas que puedas estar teniendo. Entra con tu cliente IRC en irc.freenode.net y conéctate al canal #purescript.

- El sitio web de PureScript[5] contiene enlaces a varios recursos de aprendizaje, incluyendo ejemplos de código, videos y otros recursos para principiantes.
- El repositorio de documentación de PureScript[6] contiene artículos y ejemplos de una amplia variedad de tópicos, escritos por desarrolladores y usuarios de PureScript.
- Try PureScript![7] es un sitio web que permite a los usuarios compilar código PureScript en el navegador web y contiene varios ejemplos de código simples.
- Pursuit[8] es una base de datos donde puedes buscar tipos y funciones de PureScript.

Si prefieres aprender leyendo ejemplos, las organizaciones de Github `purescript`, `purescript-node` y `purescript-contrib` contienen un montón de ejemplos de código PureScript.

Acerca del autor

Soy el desarrollador original del compilador de PureScript. Vivo en Los Angeles, California, y empecé a programar a una edad temprana en BASIC sobre un ordenador personal de 8 bits, el Amstrad CPC. Desde entonces he trabajado profesionalmente en una variedad de lenguajes de programación (incluyendo Java, Scala, C#, F#, Haskell y PureScript).

No muy tarde en mi carrera profesional, empecé a apreciar la programación funcional y sus conexiones con las matemáticas, y disfrutaba aprendiendo conceptos de programación funcional usando el lenguaje de programación Haskell.

Empecé a trabajar en el compilador de PureScript como respuesta a mi experiencia con JavaScript. Me encontré usando técnicas de programación funcional que había adquirido en lenguajes como Haskell, pero quería un entorno más escrupuloso en el que aplicarlas. Las soluciones del momento incluían varios intentos de compilar Haskell a JavaScript conservando su semántica (Fay, Haste, GHCJS), pero estaba interesado en ver si tendría éxito afrontando el problema desde el otro lado - intentando mantener la semántica de JavaScript, disfrutando la sintaxis y el sistema de tipos de un lenguaje como Haskell.

Mantengo un blog[9], y se me puede encontrar en Twitter[10].

Acerca de la traducción

Este libro es una traducción de PureScript By Example[11] de Phil Freeman[12]

[5] http://purescript.org
[6] https://github.com/purescript/documentation
[7] http://try.purescript.org
[8] http://pursuit.purescript.org
[9] http://blog.functorial.com
[10] http://twitter.com/paf31
[11] https://leanpub.com/purescript
[12] https://leanpub.com/u/paf31

Dado que muchos de los términos usados en el libro son de uso común en su forma original inglesa y sus correspondientes traducciones (buenas o malas) no lo son tanto, he incluido en el texto (donde he considerado útil) el término original en itálica y entre paréntesis. La traducción al castellano está en este repositorio GitHub[13].

Agradecimientos

Me gustaría dar las gracias a los muchos contribuyentes que ayudaron a que PureScript alcanzara su estado actual. Sin el enorme esfuerzo colectivo que se ha hecho en el compilador, herramientas, bibliotecas, documentación y pruebas, el proyecto habría fracasado sin duda.

El logo de PureScript que aparece en la portada de este libro fue creado por Gareth Hughes, y es utilizado bajo los términos de la licencia Creative Commons Attribution 4.0[14].

Finalmente, me gustaría dar las gracias a todos los que me han aportado comentarios y correcciones sobre el contenido de este libro.

[13] https://github.com/jacereda/purescript-book/tree/spanish
[14] https://creativecommons.org/licenses/by/4.0/