BBVA
Next Technologies

Fundamentos de programación funcional en JavaScript

Murcia Frontend

Febrero 2019

Índice

01

Introducción

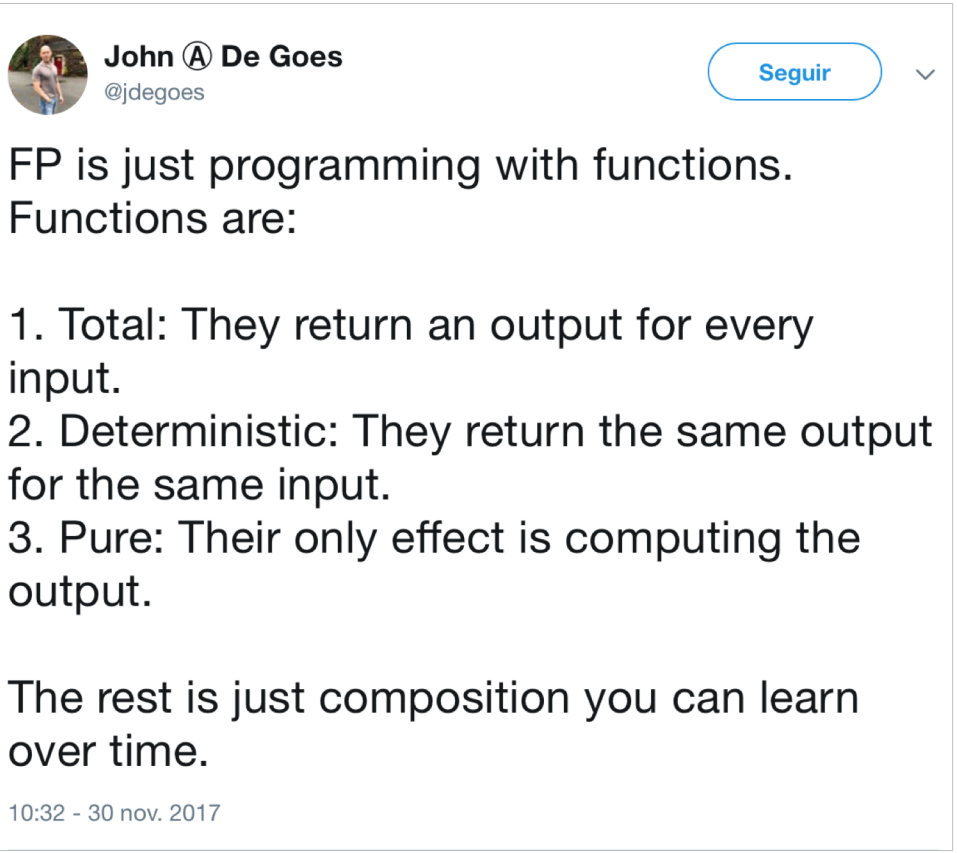
BBVA
Next Technologies

Programación funcional

FP (functional programming) es tan solo programar con funciones:

1. Totales
2. Deterministas
3. Puras

El resto es composición que puedes aprender a lo largo del tiempo



Introducción

Por qué debería importarme la programación funcional (I)

- Facilita el razonamiento y la lectura del código
- Mejora la modularidad y la reutilización de funciones
- Permite utilizar leyes y propiedades matemáticas útiles
- Tests más sencillos, menos tests necesarios
- En general, programas más extensibles, fáciles de mantener, y confiables

Introducción

Por qué debería importarme la programación funcional (y II)

	Imperativo	Funcional
Programación	Sencilla	Complicada
Razonamiento	Complejo	Simple

- Hasta ahora aprendíamos el paradigma imperativo, sencillo de programar, pero complejo de razonar
- El futuro es el paradigma funcional, a priori más complicado de programar, pero muy simple en su razonamiento

Introducción

¿Qué necesitamos para hacer programación funcional?

- Inmutabilidad

Introducción

¿Qué necesitamos para hacer programación funcional?

- Inmutabilidad
- Funciones como ciudadanos de primera clase

Introducción

¿Qué necesitamos para hacer programación funcional?

- Inmutabilidad
- Funciones como ciudadanos de primera clase
- ¡Y ya está!

Introducción

¿Qué necesitamos para hacer programación funcional?

- Inmutabilidad
- Funciones
- ¡Y ya está!

Introducción

¿Qué necesitamos para ha… funcional?

- Teoría de categorías
- Higher Kinded Types
- Álgebras e intérpretes
- …udadanos de primera clase
- ¡Y ya está!
- Typeclasses
- Tipado estático fuerte
- Etc…

Introducción

Vamos a calmarnos

- ¿Es posible hacer programación funcional en JavaScript?

Introducción

Vamos a calmarnos

- ¿Es posible hacer programación funcional en JavaScript?

02

Inmutabilidad

Inmutabilidad

¿Qué aporta la inmutabilidad?

- La mutabilidad es una fuente de comportamientos no deterministas, difíciles de seguir
- La inmutabilidad ofrece la certeza de que cuando se hace referencia a algo, ese algo no ha sido modificado jamás
- Mantiene una trazabilidad de los cambios, es como viajar en el tiempo, depurar no es un caos
- Existe una penalización al rendimiento, pero depende mucho de qué se clone

Inmutabilidad

Inmutabilidad en JavaScript (ES6)

- No existe una forma nativa segura de proteger o clonar en profundidad un objeto
  - No, `JSON.parse(JSON.stringify(objeto))` no sirve en todos los casos
  - No, `Object.assign` tampoco sirve… ni tampoco el operador spread
- Uso de objetos y arrays de primer nivel, o bien una librería de terceros, como Immutable.js
- Establecimiento de reglas de linteo

Inmutabilidad

Inmutabilidad en JavaScript (ES6)

- No existe una forma nativa segura de proteger o clonar en profundidad un objeto
  - No, `JSON.parse(JSON.stringify(objeto))` no sirve en todos los casos
  - No, `Object.assign` tampoco sirve… ni tampoco el operador spread
- Uso de objetos y arrays de primer nivel, o bien una librería de terceros, como Immutable.js
- Establecimiento de reglas de linteo
- ¡¡Dentro código!!

Inmutabilidad

Otras opciones

- Vía clausuras y clases
- Alternativas avanzadas, ópticas
  - Ejemplo: lentes, un tipo de ópticas para recuperar o actualizar un valor profundamente anidado dentro de una estructura de datos inmutable
  - No vamos a ver ningún ejemplo en esta charla, o quizás ya lo hayamos visto 😉

03

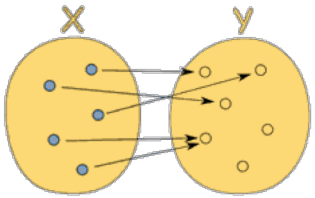
Funciones puras

BBVA
Next Technologies

Concepto de función

“El concepto es el concepto”,
Airbag (1997)

Relación entre dos valores,
una entrada y una salida

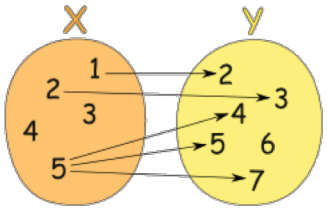


BBVA
Next Technologies

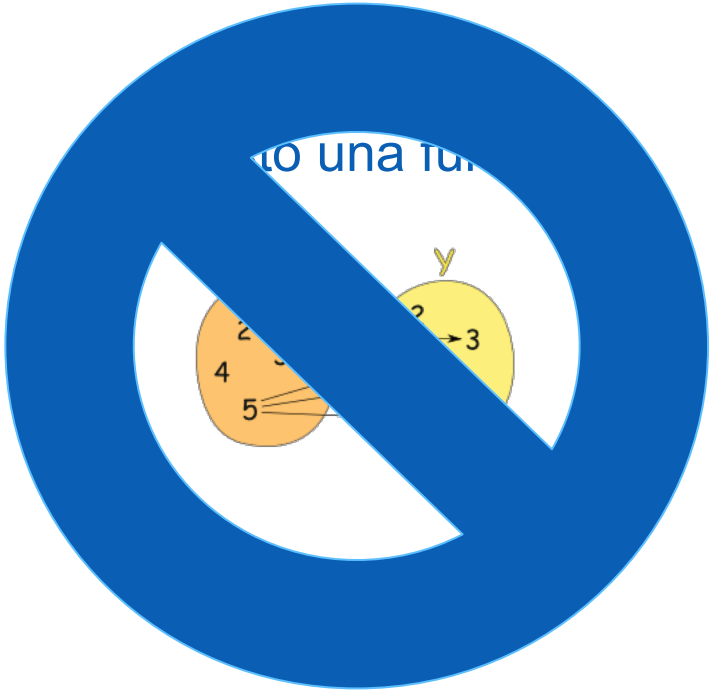
Concepto de función

“El concepto es el concepto”,
Airbag (1997)

¿Es esto una función?



BBVA
Next Technologies



Concepto de función

“El concepto es el concepto”,
Airbag (1997)

Funciones puras

Recordando de nuevo las propiedades

- Totalidad: no puede existir un valor de entrada sin un valor de salida
  - `1 / x`
- Determinismo: para el mismo valor de entrada, no puede variar el valor de la salida
  - `db.query({ select: [ 'age' ], from: 'users', where: { id: 2 });`
- Pureza: el único cometido de la función es calcular el valor de salida a partir del valor de entrada, y nada más

Funciones puras

Qué tipo de impurezas (efectos colaterales) existen

- Enviar o recibir datos por la red
- Imprimir por pantalla o leer de teclado
- Logging
- Escribir o leer de bases de datos
- Generar un número aleatorio
- En general, cualquier acción que produzca un cambio más allá de devolver un valor

Funciones puras

¿Por qué tanta insistencia con la pureza?

- Poder razonar sobre el código igual que se haría con ecuaciones matemáticas
- Las funciones puras ofrecen transparencia referencial, la clave para poder aplicar el punto anterior, siempre devuelven lo mismo
- Cacheables, más fácilmente testables, paralelizable, optimizable por el compilador… hay muchos beneficios derivados de esta propiedad

Funciones puras

¿Por qué tanta insistencia con la pureza?

- Poder razonar sobre el código igual que se haría con ecuaciones matemáticas
- Las funciones puras ofrecen transparencia referencial, la clave para poder aplicar el punto anterior, siempre devuelven lo mismo
- Cacheables, más fácilmente testables, paralelizable, optimizable por el compilador… hay muchos beneficios derivados de esta propiedad
- ¡¡Dentro código!!

Funciones puras

Pero, precisamente las funciones impuras son el código “útil”

- No queremos eliminarlas, sólo queremos llevarlas a las capas más externas de nuestra aplicación, y trabajar lo máximo posible con funciones puras

- La purificación más sencilla, no ejecutar

```
const getUser = user => () => db.query(user);
```

- La composición de funciones y algunos tipos de datos ayudan a lidiar con este tipo de impurezas

Funciones puras

Pero, precisamente las funciones impuras son el código “útil”

- No queremos eliminarlas, sólo queremos llevarlas a las capas más externas de nuestra aplicación, y trabajar lo máximo posible con funciones puras

- La purificación más sencilla, no ejecutar

```
const getUser = user => () => db.query(user);
```

- La composición de funciones y algunos tipos de datos ayudan a lidiar con este tipo de impurezas

- Promise es una construcción nativa en JavaScript para manejar la impureza “asincronía”

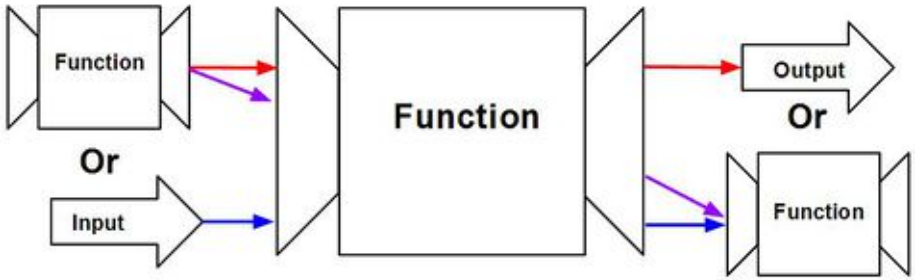
04

Funciones de orden superior

BBVA
Next Technologies

Funciones de orden superior

- Funciones que aceptan funciones como argumentos
- Funciones que devuelven funciones como salida



Funciones de orden superior

¿Para qué sirven?

- En el caso de ser argumentos de otras funciones, intuitivamente sirven para inyectar comportamientos
  - Ejemplo: predicados lógicos
- En el caso de ser la salida de otras funciones, sirven como factorías genéricas
  - Ejemplo: funciones parcialmente aplicadas
- Por cierto, no hemos hablado de las famosas lambdas… son simples funciones anónimas

Funciones de orden superior

Beneficios

- Abstracción sobre funciones como si fueran un valor más
- La función llamante obtiene control sobre parte de la ejecución de la función a la que llama (inversión de control)
- Permiten la composición de funciones
- En general, un código más reutilizable y limpio, en la línea de decir qué hacer, y no cómo hacerlo

Funciones de orden superior

¿Cómo se utilizan en JavaScript?

- El prototipo Array ya viene equipado con varias funciones de orden superior
  - map, reduce, filter, find, etc.
- Promise también tiene sus propias funciones de orden superior, como then
- Lo habitual será crear nuestras propias funciones de orden superior aceptando otras funciones como argumentos

Funciones de orden superior

¿Cómo se utilizan en JavaScript?

- El prototipo Array ya viene equipado con varias funciones de orden superior
  - map, reduce, filter, find, etc.
- Promise también tiene sus propias funciones de orden superior, como then
- Lo habitual será crear nuestras propias funciones de orden superior aceptando otras funciones como argumentos
- ¡¡Dentro código!!

05

Currificación

Currificación

No es una técnica culinaria

- Pero sí vamos a pasar nuestras funciones por una batidora para dejarlas más componibles
- En una sola frase, consiste en la transformación de funciones de múltiples argumentos en una cadena de funciones de un único argumento
- Nos va a permitir utilizar más cómodamente la definición matemática de función, donde únicamente teníamos un argumento a la entrada
- Conduce directamente al concepto de aplicación parcial

Currificación

Cómo currificar

- f: $(x_1, x_2, \ldots, x_n) \Rightarrow S$
  - se aplica con f(1, 2, 3) y devuelve S
- f: $x_1 \Rightarrow x_2 \Rightarrow \ldots \Rightarrow x_n \Rightarrow S$
  - la aplicación equivalente sería f(1)(2)(3) y devuelve S
  - pero si aplicamos con f(1) devuelve $x_2 \Rightarrow (x_3 \Rightarrow \ldots \Rightarrow x_n \Rightarrow S)$
  - en cada aplicación obtenemos $x_i \Rightarrow (x_{i+1} \Rightarrow x_{i+2} \Rightarrow \ldots \Rightarrow x_n \Rightarrow S)$

Currificación

Currificar en JavaScript

- Es muy sencillo currificar mediante la notación arrow

- Es posible programar una función de orden superior curry (y su homóloga uncurry) que currifique mecánicamente cualquier función

  - O utilizar una función de librerías de terceros

Currificación

Currificar en JavaScript

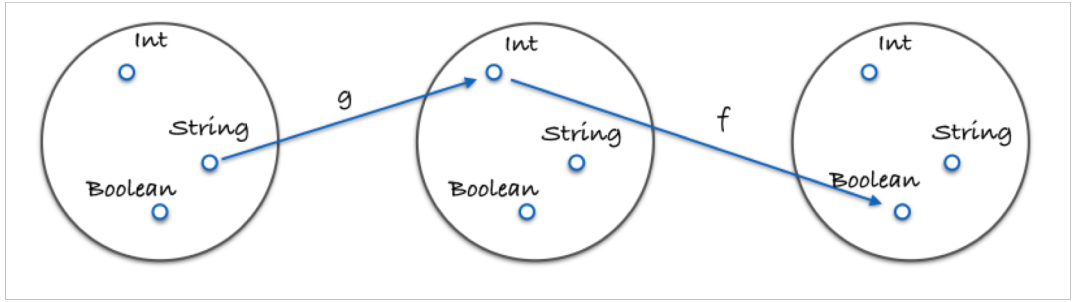
- Es muy sencillo currificar mediante la notación arrow
- Es posible programar una función de orden superior curry (y su homóloga uncurry) que currifique mecánicamente cualquier función
  - O utilizar una función de librerías de terceros
- ¡¡Dentro código!!

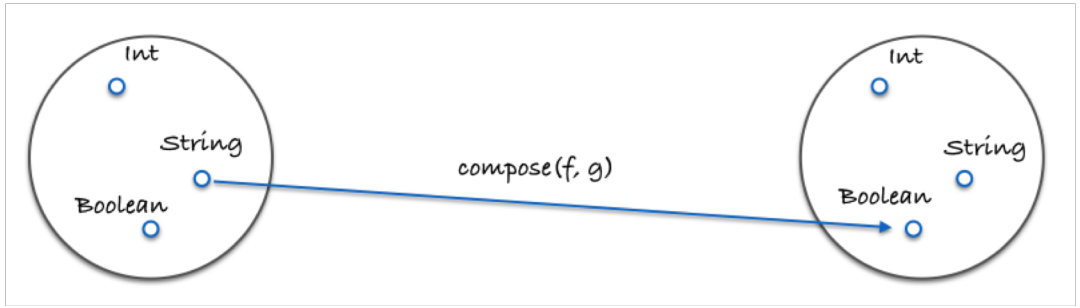
06

Composición

BBVA
Next Technologies

Composición





f(g(x)) === compose(f, g)(x)

Composición

La piedra angular de la programación funcional

- Idealmente, nuestro programa podría ser una función gigantesca compuesta de funciones más pequeñas

```
const compose = f => g => x => f(g(x));
```

- Intuitivamente como una cañería, donde la salida de una función es la entrada de la siguiente, teniendo que coincidir en tipos

- Siempre se lee de derecha a izquierda, pero es posible darle la vuelta a los argumentos, utilizando una función que habitualmente se denomina flip o swap

Composición

Propiedades de la composición

- La composición es asociativa
  - compose(f, compose(g, h)) === compose(compose(f, g), h)
- Permite derivar propiedades de otras funciones
  - compose(map(f), map(g)) === map(compose(f, g))

Composición

Estilo pointfree

- Combinación de la composición con la currificación y la aplicación parcial
- ¡Permite describir un programa sin mencionar nunca los datos!
- Usado en exceso puede conducir a ofuscación innecesaria

Composición

Estilo pointfree

- Combinación de la composición con la currificación y la aplicación parcial
- ¡Permite describir un programa sin mencionar nunca los datos!
- Usado en exceso puede conducir a ofuscación innecesaria
- ¡¡Dentro código!!

Preguntas

Consejos

Desde la experiencia

- Tener la mente abierta, al principio es difícil ver ventajas evidentes, hay una resistencia al cambio natural
- Lo más fácil es combinar programación funcional e imperativa poco a poco
  - Funciones puras
  - Estructuras de datos inmutables
  - Tipos de datos avanzados, efectos
- Es interesante estudiar álgebra de nuevo

Referencias

Artículos (I)

- Professor Frisby's Mostly Adequate Guide to Functional Programming
  - https://mostly-adequate.gitbooks.io/mostly-adequate-guide/
- Todo lo que necesitas saber de inmutabilidad en JavaScript
  - https://medium.com/dailyjs/the-state-of-immutability-169d2cd11310
- Inmutabilidad sin librerías en JavaScript
  - https://www.maxpou.fr/immutability-js-without-library

Referencias

Artículos (II)

- Todo lo que necesitas saber de funciones de orden superior
  - https://eloquentjavascript.net/05_higher_order.html
- Por qué usar tipos de datos en vez de null o undefined
  - https://oliverjash.me/2017/04/10/why-use-a-maybe-type-in-javascript
- Railway oriented programming
  - https://fsharpforfunandprofit.com/rop/

Referencias

Artículos (y III)

- Cómo cocinar un buen curry
  - https://blog.benestudio.co/currying-in-javascript-es6-540d2ad09400

- Composing software (serie de artículos sobre JavaScript funcional)
  - https://medium.com/javascript-scene/the-rise-and-fall-and-rise-of-functional-programming-composable-software-c2d91b424c8c

Enlaces

Librerías, frameworks, herramientas

- Ramda, https://ramdajs.com
- Sanctuary, https://sanctuary.js.org
- Crocks, https://evilsoft.github.io/crocks/
- FunFix, https://github.com/funfix/funfix
- Redux, https://redux.js.org/introduction/motivation

Nuestra misión es...

transformar el mundo a través de la tecnología.

Somos...

100%

BBVA Group

Más de 1.200 empleados

Nuestros pilares

Tecnología

- **> 30.000** horas al año dedicadas a la **innovación**
- **> 170** certificaciones en **cloud computing**

Construimos plataformas, servicios de seguridad, aplicaciones de negocio y productos para BBVA y otras empresas líderes

Personas

- **> 170 ponencias** de nuestros empleados en eventos top en 2017. Equipo excepcional al que le une la pasión por la tecnología.
- **>100 pizzas** hablando de tecnología en 2017. Una cultura que fomenta la formación continua

BBVA
Next Technologies

Somos expertos en...

Nuevas tecnologías

- Arquitecturas cloud avanzadas
- Big Data
- Inteligencia artificial
- Human Computer Interaction (HCI)
- Blockchain
- Automatización de procesos (BPA)

Mindset digital

- Agile coaching
- Cultura DevOps
- User Experience

Nuevos modelos de trabajo

Tecnologías de seguridad avanzada

- Soluciones de infraestructura y aplicaciones
- Desarrollo de soluciones de software seguras
- Soluciones de ciberseguridad

Partners

amazon web services · Google · Azure · redhat

Tecnologías con las que trabajamos

cloudera · Hortonworks · Spark · kafka · FOREMAN · ANSIBLE · puppet labs · Jenkins · RAML · ANGULARJS · node.js · polymer · mongoDB · neo4j