flatMap의 속사정

김나헌

흔히 봐왔던 flatMap

map + flatten

```
// [2, 3, 4, 5]
Array(1, 3).flatMap { n =>
    Array(n+1, n+2)
}
```

하지만 flatMap은 다양하다.

```
Option(nullable).flatMap { v =>
    Some(v + another)
}
```

Category Theory의 아이들

```
trait Functor[T] {
  def map[S](f: T => S): Functor[S]
}

trait Monad[T] extends Functor[T] {
  def flatMap[S](f: T => Monad[S]): Monad[S]
}
```

Container 또는 Context

interface

type T as param
=> return type S

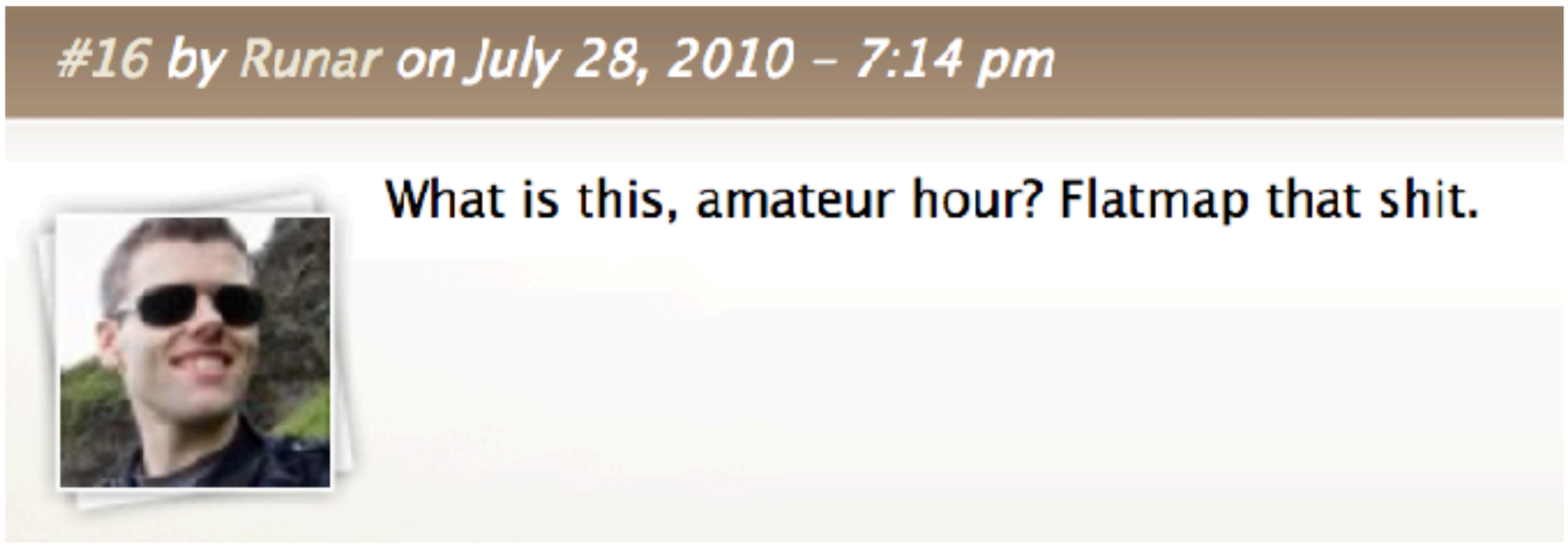
```
trait Functor[T] {
  def map[S](f: T => S): Functor[S]
}

trait Monad[T] extends Functor[T] {
  def flatMap[S](f: T => Monad[S]): Monad[S]
}
```

* T는 Monad인 어떤 타입. Array, Option, Try, …
  즉 Monad는 모든 가능한 T의 **공통적인 속성**
* 현재 Context로부터 다음 Context를 생성하는 방식을 결정
  명령문 사이의 상태 전파의 순수 함수적 형태


monad
array
option
future

flatMap의 속사정을 이해한다면, 이제



Cédric Beust, Why Scala's "Option" and Haskell's "Maybe" types won't save you from null
http://web.archive.org/web/20130512232946/http://beust.com/weblog/2010/07/28/why-scalas-option-and-haskells-maybe-types-wont-save-you-from-null/

명령형 코드를 더 우아하고 안전하게

```
val option = Option(nullable)
if (option.isDefined) {
  Some(option.get + another)
} else {
  None
}
```

option 상태 전파

다른 변수와 결속

option 으로부터
새 변수 생성

```
Option(nullable).flatMap {
  v => Some(v + another)
}
```

합성, 혹은 벽돌쌓기

고수준 언어의 핵심은 그것이 당신에게 **더 많은 추상화, 즉 더 커다란 벽돌을 제공**함으로써 당신이 **벽을 쌓을 때 필요한 벽돌의 개수를 줄여주는 것**이다. 그래서 언어가 더 강력 할수록, 프로그램은 더 짧아진다. (글자의 수를 말하는 것이 아니라 서로 구별되는 요소의 수를 말하는 것이다.)

- 해커와 화가, 폴 그레이엄

추상 위의 추상

추상 리팩토링 부터 공통화, 라이브러리, 플랫폼 까지…
수많은 코드와 개발자가 지향하는 방향

함수형 패러다임을 학습한다는 것
~~람다, 함수 체인을 사용할 수 있다~~
추상을 내제화하는 과정


Maybe we can iterate over...
flatMap that shit!