6 Haskell 重要编程模式——Functor,Applicative Functor,Monad

6.1 Functor

6.1.1 Functor的基本概念

Functor 是一个类型类(type class),先看它是如何实现的，然后通过一些例子来体会什么是Functor以及为什么要抽象出这样的概念。

```
class Functor f where
    fmap :: (a -> b) -> f a -> f b
```

这里的a,b是类型参数(type variable),f是类型构造符(type constructor),和前面讲的值构造符(value constructor) 类似，(f a)就构成了一种新的类型。比如f是Maybe,a就可以是Int,Float,etc.(Maybe Int),(Maybe Float)这些就是新 类型了。 看到fmap,很自然地会想起一个函数，map

```
map :: (a -> b) -> [a] -> [b]
```

比较一下fmap和map就会发现他们很像，只需要把f a 换成[a]就行了，刚才已经说了f a是一种类型，以a为类型参数， 如果我们让f为List, 那f a 就是List a,把List a写成[a]的形式,famp就被实例化成了map，即map只不过是fmap的特例。 这也就是说List是Functor的一个实例(instance),即

```
instance Functor [] where
    fmap = map
```

List作为Functor的实例，必须实现fmap,而List的fmap就是map函数。

```
ghci>fmap (*2) [1,2,3]
[2,4,6]
ghci>map (*2) [1,2,3]
[2,4,6]
```

6.1.2 Functor实例——Maybe

先来介绍一下著名的Maybe,Maybe的定义是

```
data Maybe a = Just a | Nothing
     deriving (Eq, Ord)
```

为什么要引入Maybe的概念呢，因为有时候，同一个函数的返回值，可能是一种错误，也可能是具体的值，比如 浮点值，这时候就可以把函数的返回值设置成Maybe Float

```
mDivide :: Float -> Float -> Maybe Float
mDivide _ 0 = Nothing
mDivide x y = Just (x/y)
ghci>mDivide 4.3 0
Nothing
ghci>mDivide 4.3 2
Just 2.15
```

如果把Maybe就是Functor的一个实例,我们就可以将函数作用到Maybe类型的值上

```
instance Functor Maybe where
    fmap f (Just x) = Just (f x)
    fmap f Nothing = Nothing
ghci>fmap (+1) (Just 3)
Just 4
ghci>fmap (+1) Nothing
Nothing
```

从这里我们也可以看出Functor的作用，如果一个类型由类型构造符(type constructor)如Maybe,List 和具体类型如Int,Float等等组成，那么就可能把这个类型构造符作为Functor的实例，从而定义当一个 函数作用到这个类型上时，类型的值如何变化。

6.2 Applicative Functor

6.2.1 Applicative Functor基本概念

Applicative Functor是什么？顾名思义，它是一个Functor，而且是一个更为强大的Functor。Appllicative 的定义在Control.Applicative中：

```
class (Functor f) => Applicative f where
    pure :: a -> f a
    (<*>) :: f (a -> b) -> f a -> f b
```

从定义可以看出，Applicative是一个类型类(type class),即如果一个类型是Appllicative的实例，那么我们可以在 对这个类型进行Applicative实例化的时候，定义这个类型的一些行为方式。这些行为方式的定义即是对pure和<*>的实现。 就像在Functor里对fmap的实现一样。 另外，定义里还有一个类型限制，即需要实例化的f必须首先是Functor的实例。 pure接收一个值，返回一个Applicative值，比如接收一个Int类型的值，返回一个Maybe Int类型的值。 <*>接收一个functor 值，即f (a->b)，再接收一个functor 值，然后从第一个functor里抽出一个函数，作用在第二个 functor的值上，最后得到另一个functor。 现在先大概了解一个pure和<*>的基本概念，以后遇到的例子多了，自然会理解。

6.2.2 Applicative Functor实例——Maybe

先看看Maybe作为Applicative Functor实例的定义：

```
instance Applicative Maybe where
    pure = Just
    Nothing <*> _ = Nothing
    (Just f) <*> something = fmap f something
```

看到例子可能会理解了pure和<*>到底是干什么的，也理解了为什么Maybe需要是Functor的实例。 pure = Just 说明什么呢？pure 3 = Just 3,看看pure 类型定义， a -> f a,这里的a 是Int,f 是Maybe。 pure定义了如何由一个普通值转化为一个Applicative值。 对于<*>的定义中出现了fmap,回忆一下fmap的定义，会对<*>的作用多一点了解。

```
class Functor f where
```

```
    fmap :: (a -> b) -> f a -> f b
```

举例来说，(Just f) <*> (Just a) = fmap f (Just a) = (Just (f a))

```
ghci>(Just (+1)) <*> (Just 3)
Just 4
ghci>fmap (+1) (Just 3)
Just 4
```

另外，由于pure = Just,还可以写成以下形式

```
ghci>(pure (+1)) <*> (Just 3)
Just 4
```

6.2.3 Applicative 风格

另外，我们还可以这样玩

```
ghci>pure (+) <*> Just 1 <*> Just 3
Just 4
ghci>pure (+) <*> Just 1 <*> Nothing
Nothing
ghci>pure (+) <*> Nothing <*> Just 3
Nothing
```

这又是什么意思呢？从下面的推导过程可以看出最后的结果是怎么得来的。

```
pure (+) <*> Just 1 <*> Just 3
= Just (+) <*> Just 1 <*> Just 3
= (Just (+) <*> Just 1) <*> Just 3
= (fmap (+) Just 1) <*> Just 3
= Just (+1) <*> Just 3
= fmap (+1) Just 3
```

```
= Just fmap (+1) 3
= Just 4
```

6.3 Monad

6.3.1 Monad的基本概念

Monad 像Functor,Applicative一样，也是一个类型类(type class),那么，它是如何定义的呢？

```
class Monad m where
    return :: a -> m a

    (>>=) :: m a -> (a -> m b) -> m b

    (>>) :: m a -> m b -> m b
    x >> y = x >>= \_ -> y

    fail :: String -> m a
    fail msg = error msg
```

又像Functor,Applicative一样，成为Monad的实例，需要定义一些函数，以决定实例的一些行为。 第一个函数是return,它不同于C,C++,Java等语言中的return,这里的return和Applicative里的pure一样， 它其实是把一个普通值转化成了一个monad值，例如5转化成Just 5。 第二个函数是(>>=),看它的类型，是个非常奇怪的函数。第一个参数是一个monad值，第二个参数一个函数， 从monad值里提取出一个值，把这个值作为函数（第二个参数）的参数，最后得到另一个monad值。

6.3.2 Monad实例——Maybe

先看看定义：

```
instance Monad Maybe where
    return x = Just x
    Nothing >>= f = Nothing
    Just x >>= f  = f x
```

```
    fail _ = Nothing
```

return 的定义方式和Maybe作为Applicative实例时pure的定义方式一样。 >>= 的定义正如刚才讲的，从Just x里提取出x,作为f的参数。 这只是Monad的一个简单概念，Monad在Haskell里有许多应用，需要在实际中去应用。