FP and Rx

函数式编程和 Rx 简明介绍

https://github.com/nanomichael

一个例子

```
if (a != null) {
  b = getB(a);
  if (b != null) {
    c = getC(b);
    if (c != null) {
      d = getD(c);
      // ...
    }
  }
}
```

Optional and `'?.'`

```
Optional
  .ofNullable(a)
  .flatMap(a -> Optional.ofNullable(getB(a)))
  .flatMap(b -> Optional.ofNullable(getC(b)))
  .flatMap(c -> Optional.ofNullable(getD(c)))
  // ...
  .ifPresent(x -> doSomething(x));
```

```
a
  ?.let { getB(it) }
  ?.let { getC(it) }
  ?.let { getD(it) }
  // ...
  ?.run { doSomething(it) }
```

另一个例子

```
val a = listOf(1, 2, 3)
val b = listOf(4, 5, 6)
val c = listOf(7, 8, 9)
// ...
val list = mutableListOf<List<Int>>()
for (ai in a) {
  for (bi in b) {
    for (ci in c) {
      // ...
      list += listOf(ai, bi, ci, ...)
    }
  }
}
```

flatMap

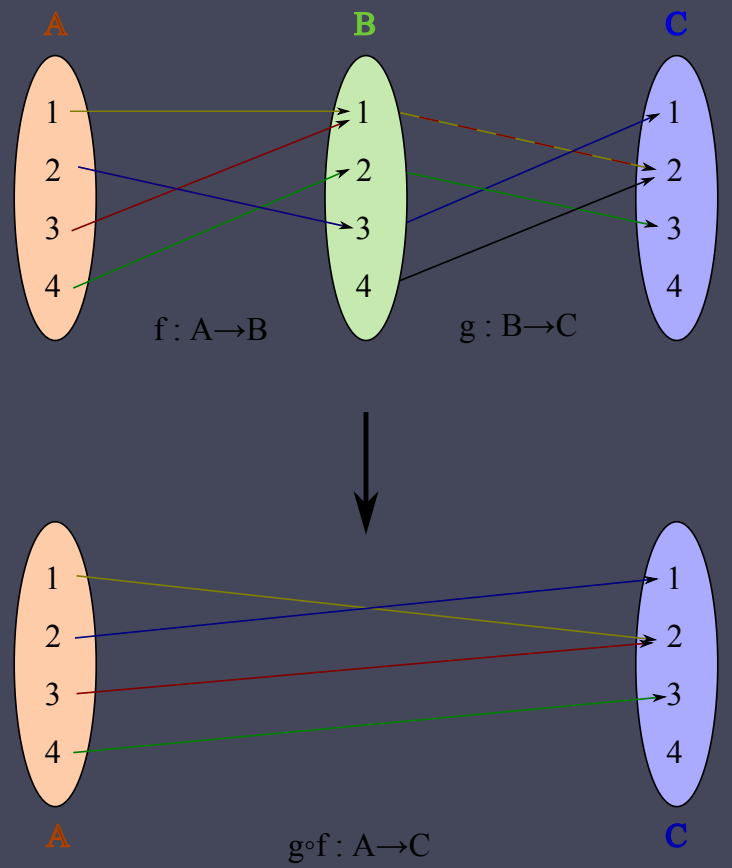
```
val list = a
  .flatMap { ai ->
    b.map { bi ->
      listOf(ai, bi)
    }
  }
  .flatMap { list ->
    c.map { ci ->
      list + ci
    }
  }
  // ...
```

将 `if` 和 `for` 后的代码块看作是回调函数，同步代码也会产生 `callback hell`。

Function

> In mathematics, a function is a binary relation over two sets that associates every element of the first set, to exactly one element of the second set.

- 一个集合到另一个集合的映射
- 给定一个输入，有且仅有一个输出
- 函数间可组合，记作：`(f.g)(x) = f(g(x))`



Category theory

> In math-speak, categories are:
>
> 1. collections of "objects" (you should think of sets),
> 2. and "arrows" (you should think of functions between sets),
> 3. where each arrow has a domain and a range,
> 4. each object has an "identity" arrow (think of the identity function, where `f(x) = x`)
> 5. and arrows can be composed when the domains and ranges match up right.
>
> -- *Why do monads matter?*

Function composition

> Functional programming is all about function composition, nothing more.

一个例子

给定一个文件，内容为一个网络地址：

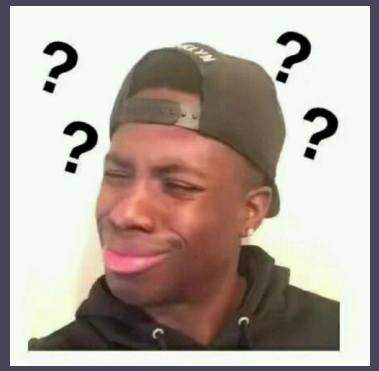
- 读取文件内容
- 将文件内容作为参数，获取网络资源
- 保存内容到本地并返回文件地址

```
fun readFile(path: String): String = // ...
fun readNet(url: String): String = // ...
fun saveIntoFile(content: String): String = // ...

val contentPath = saveIntoFile(readNet(readFile(filePath)))
```

另一种形式

```
(f . g) x = f g x
```

```
infix fun <T, R1, R2> ((R1) -> R2).compose(g: (T) -> R1) = { t: T ->
  this(g(t))
}

val f = ::saveIntoFile compose ::readNet compose ::readFile
val contentPath = f(filePath)
```



Chain style

```
infix fun <T, R1, R2> ((T) -> R1).then(f: (R1) -> R2) = { t: T ->
  f(this(t))
}

val f = ::readFile then ::readNet then ::saveIntoFile
val contentPath = f(filePath)
```

Async

```
fun readFileCallback(path: String, callback: (url: String) -> Unit) {
  callback(readFile(String))
}
fun readNetCallback(url: String, callback: (content: String) -> Unit) {
  callback(readNet(String))
}
fun saveIntoFileCallback(content: String, callback: (path: String) -> Unit) {
  callback(saveIntoFile(content))
}

readFileCallback(path) { url ->
  readNetCallback(url) { content ->
    saveIntoFileCallback(content) { path -> /* do something */ }
  }
}
```

我们希望的是：

```
val readFileF = ???
val readNetF = ???
val saveIntoFileF = ???

val f = readFileF then readNetF then saveIntoFileF
f { path ->
  doSomething()
}
```

各个异步回调函数也能进行组合

Currying

所谓「柯里化」，就是将一个有多个参数的函数转换为多个只有一个参数的函数。编程语言 `Haskell` 天然支持「柯里化」。

```
sum :: (Num a) -> a -> a -> a
sum x y = x + y
let sum3 = sum 3 -- 相当于 3 + ?
sum3 2 -- 相当于 3 + 2
```

Kotlin 模拟下：

```
fun <T1, T2, R> ((T1, T2) -> R).currying() = { t1: T2 ->
  { t2: T2 -> this(t1, t2) }
}
```

定义一种 `伪函数`，模拟「柯里化」，消除实际参数，只保留 `callback` 参数

```
interface F<T> {
  operator fun invoke(callback: (T) -> Unit)
}

val readFileF = object : F<String> {
  override fun invoke(callback: (String) -> Unit) {
    callback(readFile(path))
  }
}
readFileF(::println)
```

这种 `伪函数` 应可以和普通函数组合，定义为 `map`

```
fun <T, R> F<T>.map(f: (T) -> R): F<R> = object : F<R> {
  override fun invoke(callback: (R) -> Unit) {
    this@map { t -> callback(f(t)) }
  }
}
```

还应有 `id` 函数，返回自己本身

```
class Id<T>(val t: T) : F<T> {
  override fun invoke(callback: (T) -> Unit) {
    callback(t)
  }
}
```

现在可以以 `callback` 的方式对普通函数进行链式调用了

```
val f = Id(filePath)
  .map { path -> readFile(path) }
  .map { url -> readNet(url) }
  .map { content -> saveIntoFile(content) }
f { println(it) }
```

WAO! `callback hell` 没了!

伪函数应可以和伪函数组合

Monad

FP vs Rx

Coroutine

Use Rx

- Control driven and Data driven
- Proactive and Reactive
- Purity

References