Funktionale Architektur mit Kotlin

Michael Sperber & Benedikt Stemmildt

Created: 2022-07-05 Tue 13:25

Funktionale Architektur mit Kotlin

Mike Sperber

Active Group GmbH

`@sperbsen`

Benedikt Stemmildt

BLUME2000

`@slashBene`

Funktionale Programmierung

BLUME2000 2020 - Schulung "Funktionale Programmierung"

validation.rkt - DrRacket

validation.rkt (define ...) Check Syntax Step Run Stop

```
69  ; - eine Beschreibung des Fehlers
70  (define-record (failure-of description)
71    make-failure
72    failure?
73    (failure-descriptions (list-of description))) ; verallgemeinerbar auf beliebige Monoiden
74
75  ; Ein Erfolg hat folgende Eigenschaft:
76  ; - Ergebnis
77  (define-record (success-of result)
78    make-success
79    success?
80    (success-result result))
81
82  ; Eine Validierung ist eins der folgenden:
83  ; - Fehlschlag
84  ; - Erfolg
85  (define validation
86    (lambda (description result)
87      (signature
88       (mixed (failure-of description)
89              (success-of result)))))
90
91  ; Funktor für validation
92  (: validation-map ((%a -> %b) (validation %description %a) -> (validation %description %b)))
```

Schreibe Dein Programm! 92:35 437.53 MB

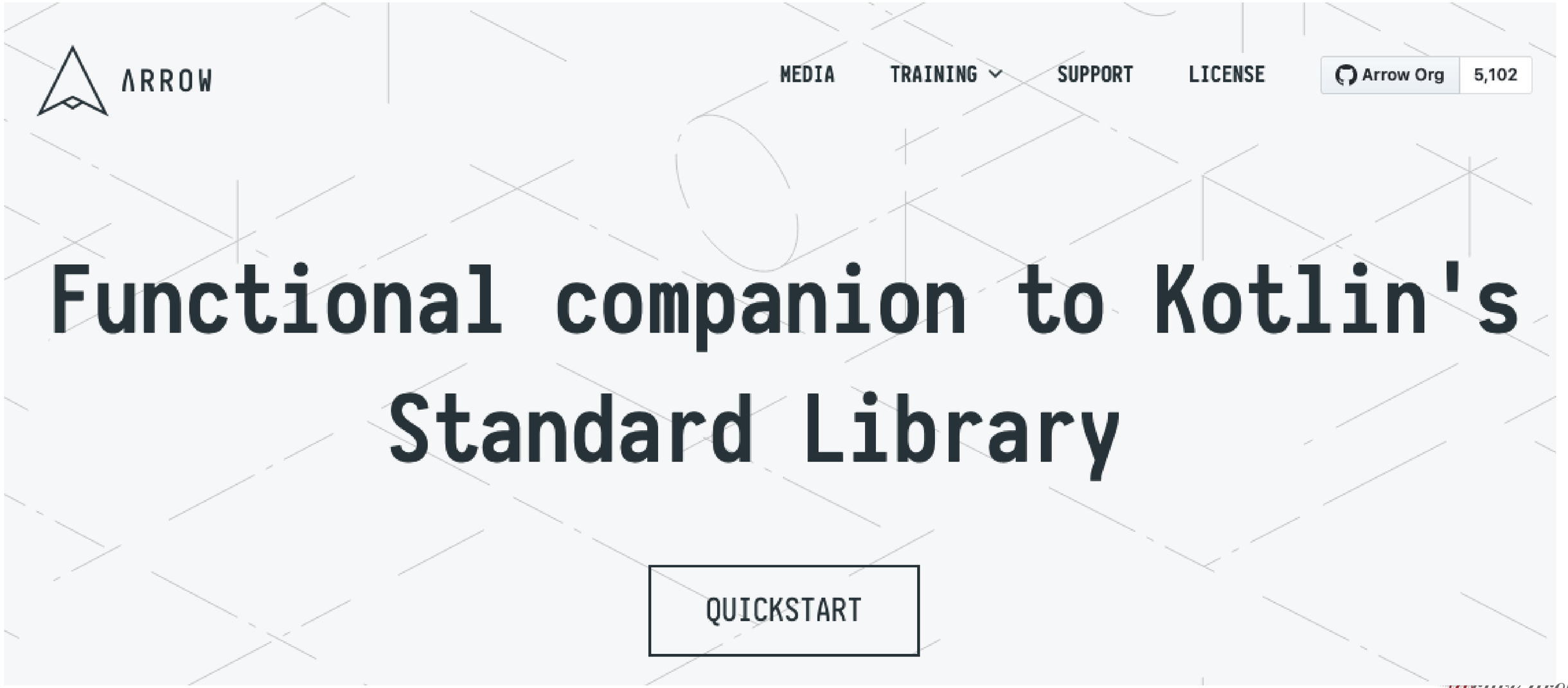
@active group

BLUME 2000

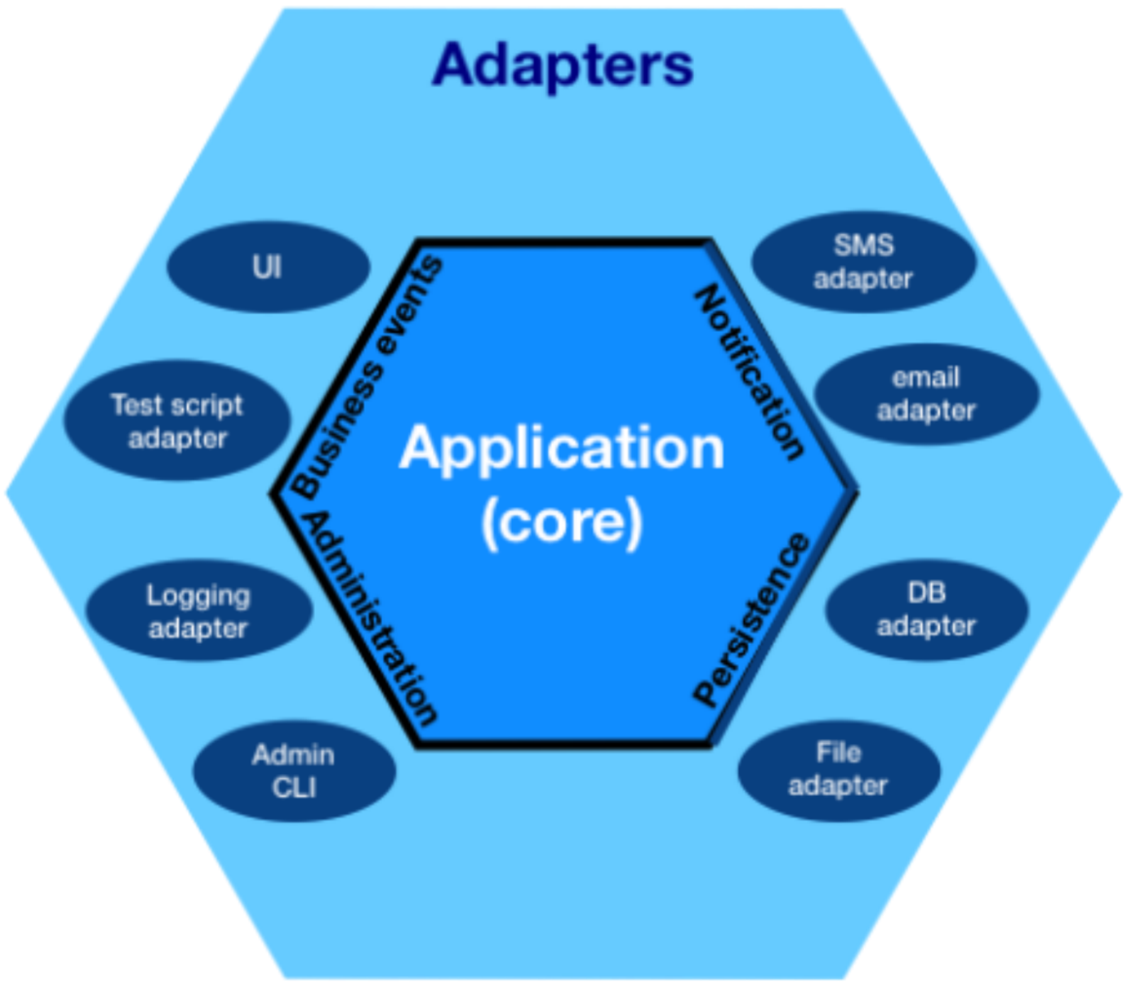
BLUME2000 2020 - Realität

Functional Programming in Kotlin?

- Lists / Option
- Validation
- Typystem



Hexagonale Architektur



Source: Wikimedia Commons, CC-SA 4.0

Profiling

```
@SuppressWarnings("ThrowsCount", "LongMethod")
  fun updateWarenkorb(
    httpWarenkorbVeraenderung: HttpWarenkorbVeraenderung,
    sessionIdPayload: String,
  ): Warenkorb {
    val start = System.currentTimeMillis()

    val produkt = produktRepository.holeProduktViaProduktNummer(Pro
    ....

val getProduktEnd = System.currentTimeMillis()
logger.info { "get Produkt from DB took ${getProduktEnd - start

if (produkt == null) { ... }
...
val warenkorbVeraenderungTransformationEnd = System.currentTimeM
logger.info { "transform into warenkorb-veränderung took ${waren
...
val warenkorbVeraenderungValidationEnd = System.currentTimeMill
logger.info { "warenkorb-veraenderung validation took ${warenko
...
val getWarenkorbEnd = System.currentTimeMillis()
```

Monaden

https://funktionale-programmierung.de/2013/04/18/haskell-monaden.html

Monaden in Kotlin

```
sealed interface ProductM<out A> {
  data class FindAllProducts<out A>
                (val callback: (Flow<Product>) -> ProductM<A>)
      : ProductM<A>
  data class FindProductById<out A>
                (val id: ProductId,
                 val callback: (Option<Product>) -> ProductM<A>)
      : ProductM<A>
  data class CountProducts<out A>
                (val callback: (Long) -> ProductM<A>)
      : ProductM<A>
  data class CreateProduct<out A>
                (val product: Product,
                 val callback: (Unit) -> ProductM<A>)
      : ProductM<A>
  data class Pure<out A>(val result: A) : ProductM<A>
}
```

Monaden benutzen

```
CreateProduct(product1, {
CreateProduct(product2, {
FindAllProducts({ products ->
   ...
})})})
```

Kotlin ist nicht Java

```
CreateProduct(product1) {
CreateProduct(product2) {
FindAllProducts { products ->
   ...
}}}
```

Programmbausteine

```
sealed interface ProductM<out A> {

  companion object {
    fun findAllProducts(): ProductM<Flow<Product>> =
        FindAllProducts(::Pure)
    fun findProductById(id: ProductId): ProductM<Option<Product>> =
        FindProductById(id, ::Pure)
    fun countProducts(): ProductM<Long> =
        CountProducts(::Pure)
    fun createProduct(product: Product): ProductM<Unit> =
        CreateProduct(product, ::Pure)
    fun <A> pure(result: A) = Pure(result)
  }

  fun <B> bind(next: (A) -> ProductM<B>): ProductM<B>
}
```

Programmbausteine zusammensetzen

```
val c1 = createProduct(product1)
val c2 = createProduct(product2)

c1.bind {
c2.bind {
findAllProducts.bind { products ->
  ...
}}}
```

Vorher

```
CreateProduct(product1) {
CreateProduct(product2) {
FindAllProducts { products ->
   ...
}}}
```

Coroutinen und Continuations

```
sealed interface ProductM<out A> {

  suspend fun susp(): A =
    suspendCoroutine { cocont: Continuation<A> ->
      val element = cocont.context[ProductCE]!! as ProductCE<A>
      element.productM = some(
        bind { result ->
          cocont.resume(result)
          element.productM.get()
        }
      )
    }
}
```

Coroutinen und Continuations

```
sealed interface ProductM<out A> {

  companion object {
   fun <A> productM(context0: CoroutineContext = EmptyCoroutineConte
                    block: suspend ProductMCoroutineDsl.() -> A)
       : ProductM<A> {
      val element = ProductCE<A>(none())
      val context = context0 + element
      val coroutine: suspend () -> A = { ProductMCoroutineDsl().bloc
      coroutine.startCoroutine(
        Continuation(context) { result ->
          result.onFailure { exception ->
            val currentThread = Thread.currentThread()
            currentThread.uncaughtExceptionHandler.uncaughtExceptio
          }
        }
      )
      return element.productM.get()
    }
  }
}
```

Coroutine

```
productM {
  createProduct(product1).susp()
  createProduct(product2).susp()
  val products = findAllProducts().susp()
  ...
}
```

DSL

```
class ProductMCoroutineDsl {
  suspend fun findAllProducts() =
      ProductM.findAllProducts().susp()
  suspend fun findProductById(id: ProductId) =
      ProductM.findProductById(id).susp()
  suspend fun countProducts() =
      ProductM.countProducts().susp()
  suspend fun createProduct(product: Product) =
      ProductM.createProduct(product).susp()

  suspend fun <A> pure(result: A): A = ProductM.pureM(result)
}
```

DSL

```
productM {
  createProduct(product1)
  createProduct(product2)
  val products = findAllProducts()
  ...
}
```

Was ist mit dem Profiling?

Läuft

```
override tailrec suspend fun <A>
  run(productM: ProductM<A>,
      db: MutableMap<ProductId, Product>): A =
  when (productM) {
    is FindAllProducts ->
      run(productM.callback(db.values.asFlow()), db)
    is FindProductById ->
      run(productM.callback(Option.fromNullable(db[productM.id])), d
    is CountProducts ->
      run(productM.callback(db.size.toLong()), db)
    is CreateProduct -> {
      db[productM.product.id] = productM.product
      run(productM.callback(Unit), db)
    }
    is Pure -> productM.result
  }
```

Dependency Injection

```
interface UnsafeProductMRunner {
  suspend fun <A> run(productM: ProductM<A>): A
}

class InMemoryProductM(val db: MutableMap<ProductId, Product>)
  : UnsafeProductMRunner

final class MongoProductM(val mongo: ReactiveFluentMongoOperations)
  : UnsafeProductMRunner
```

Was ist mit dem Profiling?

```
data class ProfilingRecord(val opSummary: String, val millis: Long)

class ProfilingRecorder(var records: MutableList<ProfilingRecord>)
  var then: Long = -1
  lateinit var summary: String

  fun opStarted(summary: String) {
    val now = System.currentTimeMillis()
    if (then != -1L)
      records.add(ProfilingRecord(this.summary, now - then))
    this.summary = summary
  }
}
```

Profiling-Runner

```
class ProfilingInMemoryProductM(val db: MutableMap<ProductId, Produc
                                val recorder: ProfilingRecorder)
  : UnsafeProductMRunner {
  override tailrec suspend fun <A> run(productM: ProductM<A>): A {
    recorder.opStarted(productM.summary())
    return when (productM) {
      is FindAllProducts ->
        run(productM.callback(db.values.asFlow()))
          . . .
      }
    }
  }
```

Separation of Concerns

```
class ProfilingProductM(val db: MutableMap<ProductId, Product>,
                        val recorder: ProfilingRecorder)
  : InMemoryProductM(db) {

  override suspend fun <A> run(productM: ProductM<A>): A {
    recorder.opStarted(productM.summary())
    return super.run(productM)
  }
}
```

- ☹ `InMemoryProductM` **muss** `open` **sein**
- ☹ **Was ist mit dem** `tailrec`**?**

Profiling-Transformation

```
fun <A> profile(productM: ProductM<A>, recorder: ProfilingRecorder)
    : ProductM<A> =
  when (productM) {
    is FindAllProducts -> {
      recorder.opStarted(productM.summary())
      FindAllProducts() { products ->
        profile(productM.callback(products), recorder)
      }
    }
    is FindProductById -> {
      recorder.opStarted(productM.summary())
      FindProductById(productM.id) { oProduct ->
        profile(productM.callback(oProduct), recorder)
      }
    }
    ...
}
```

Spring

```
@SpringBootApplication
class ExampleApplicationOne {
  @Bean
  fun outRunner(mongo: ReactiveFluentMongoOperations,
                @Value("\${spring.kafka.bootstrap-servers}")
                  kafkaBootstrapServers: String)
      : UnsafeProductMRunner {
    val impl =
      KafkaProducerProductMDecorator(
        bootstrapAddress = kafkaBootstrapServers,
        delegate = KafkaConsumerProductMDecorator(
          bootstrapAddress = kafkaBootstrapServers,
          delegate = MongoProductMDecorator(mongo = mongo)
        )
      )
    return ImplementationProductMRunner(impl)
  }
  ...
}
```

Zusammenfassung

- FP kann OO/hexagonale Architektur verbessern
- Kotlin + FP = ❤️
- Monaden FTW
- Effekt-Kombination mit Decorator-Pattern
- funktionale Sprachen + FP = ❤️❤️❤️
- "proper tail calls" fehlen immer noch auf der JVM

https://gitlab.com/BeneStem/verticalization-example-service-one

iSAQB-Community-Treffen, Stuttgart

25.7.2022, 18:00, Kulturkiosk

https://www.meetup.com/isaqb-community/events/286631012/