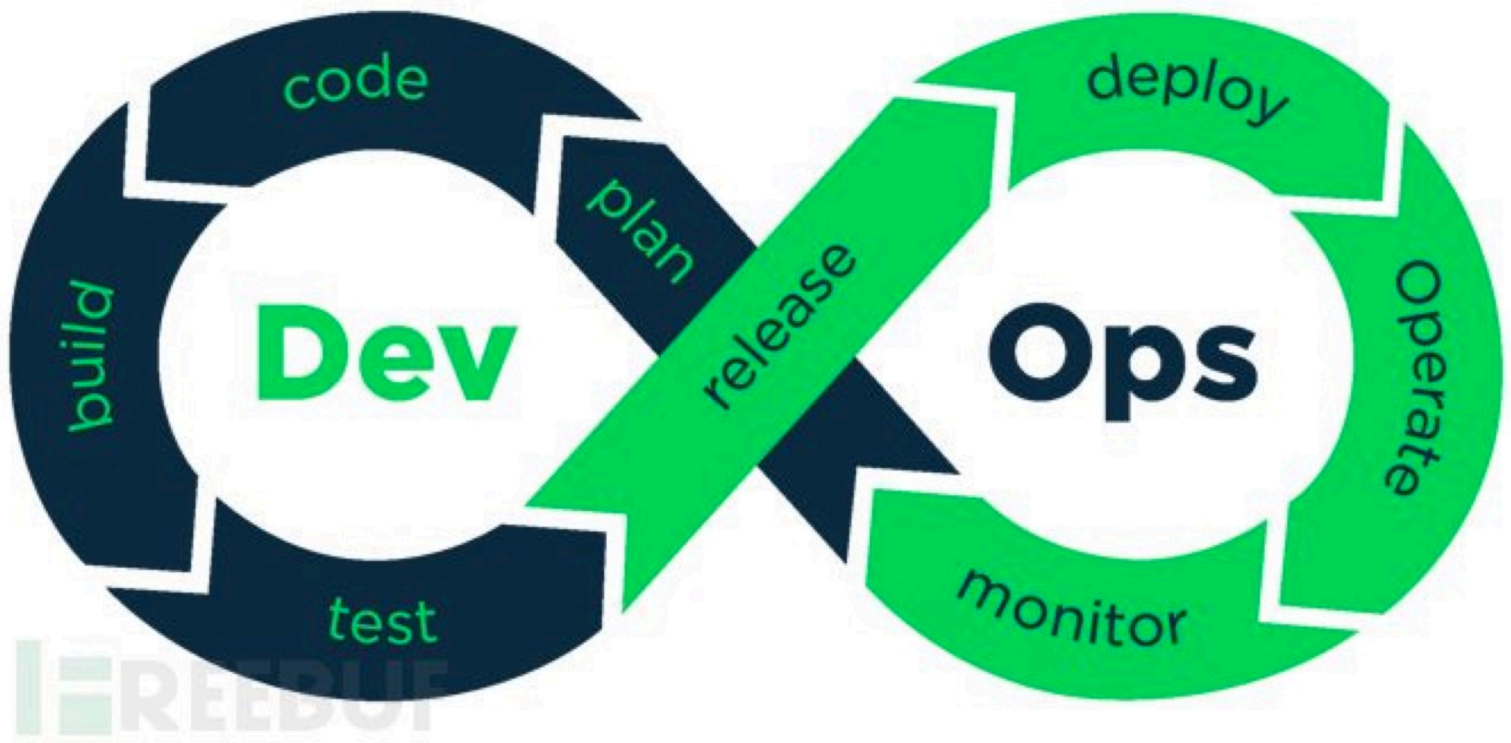
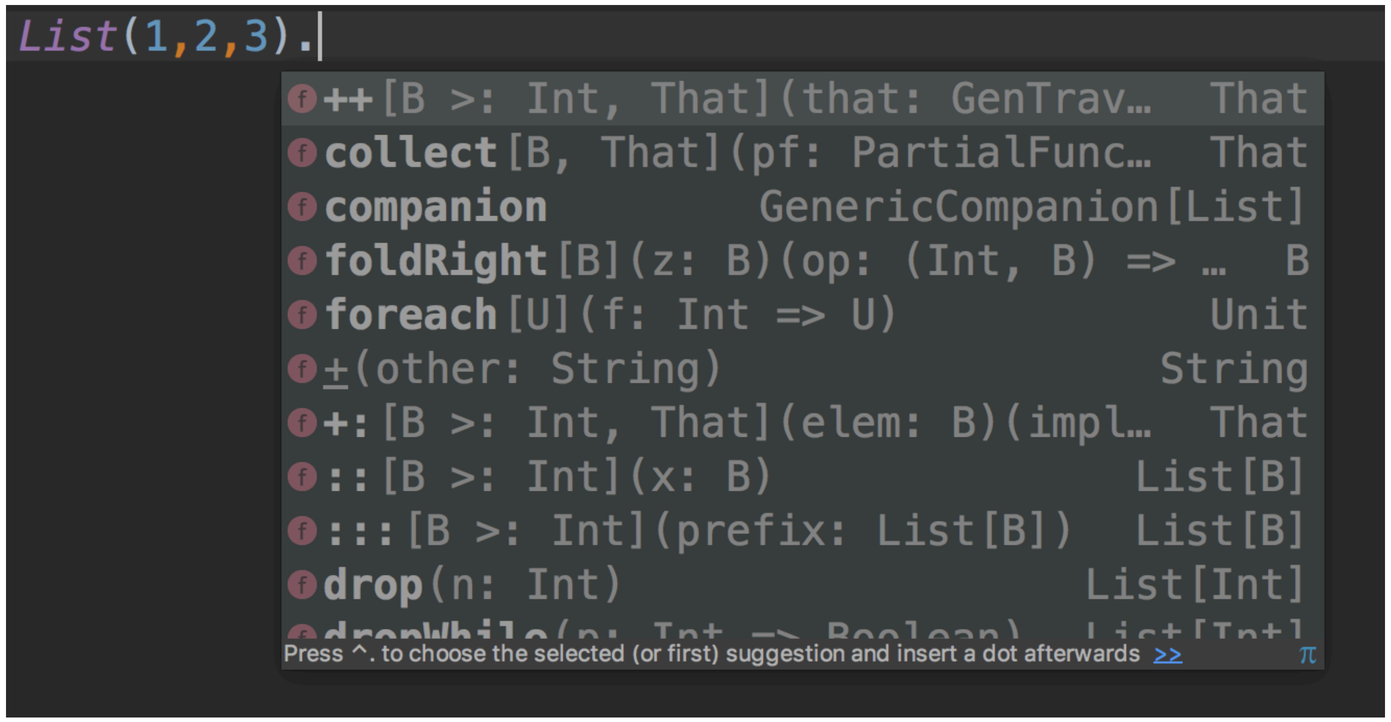
ThoughtWorks®

DDD是一种纪律

杨云（大魔头）

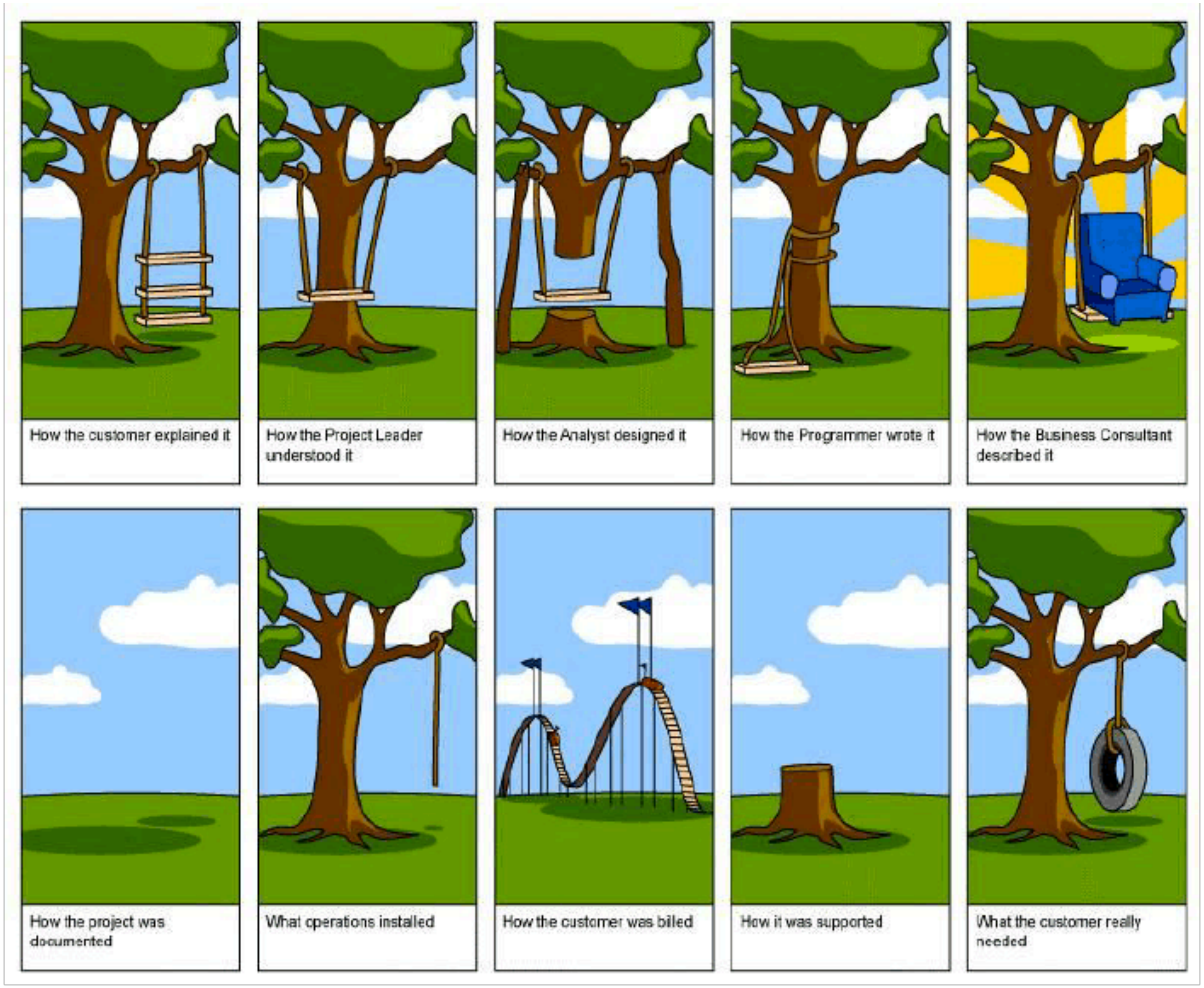
自动化正在吞噬一切





```
List(1,2,3).
  ++[B >: Int, That](that: GenTrav…   That
  collect[B, That](pf: PartialFunc…   That
  companion            GenericCompanion[List]
  foldRight[B](z: B)(op: (Int, B) => …    B
  foreach[U](f: Int => U)              Unit
  +(other: String)                   String
  +:[B >: Int, That](elem: B)(impl…   That
  ::[B >: Int](x: B)                List[B]
  :::[B >: Int](prefix: List[B])    List[B]
  drop(n: Int)                    List[Int]
  dropWhile(p: Int => Boolean)    List[Int]
Press ^. to choose the selected (or first) suggestion and insert a dot afterwards >>   π
```

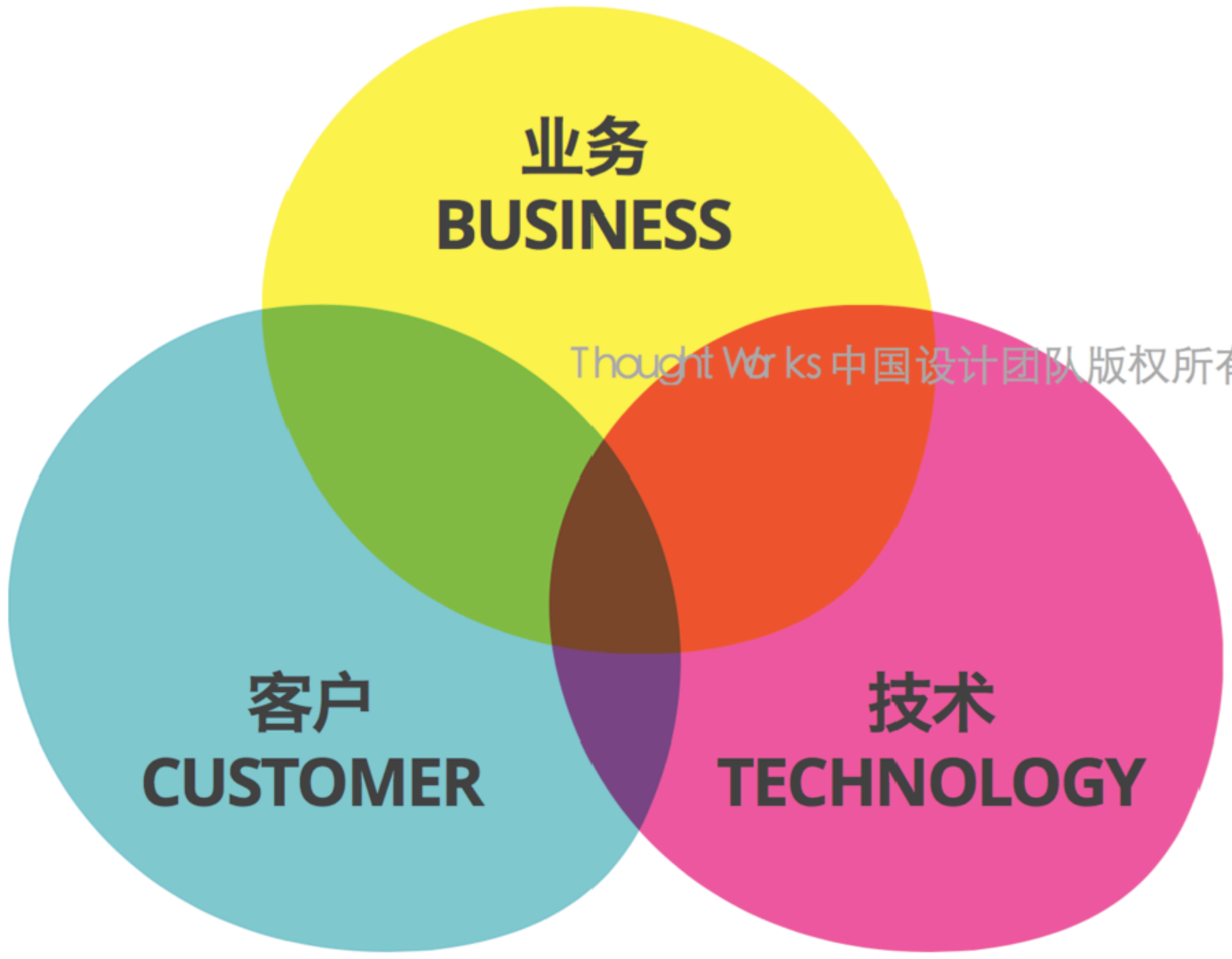
幸好一件事无法被自动化（暂时）


How the customer explained it
How the Project Leader understood it
How the Analyst designed it
How the Programmer wrote it
How the Business Consultant described it
How the project was documented
What operations installed
How the customer was billed
How it was supported
What the customer really needed

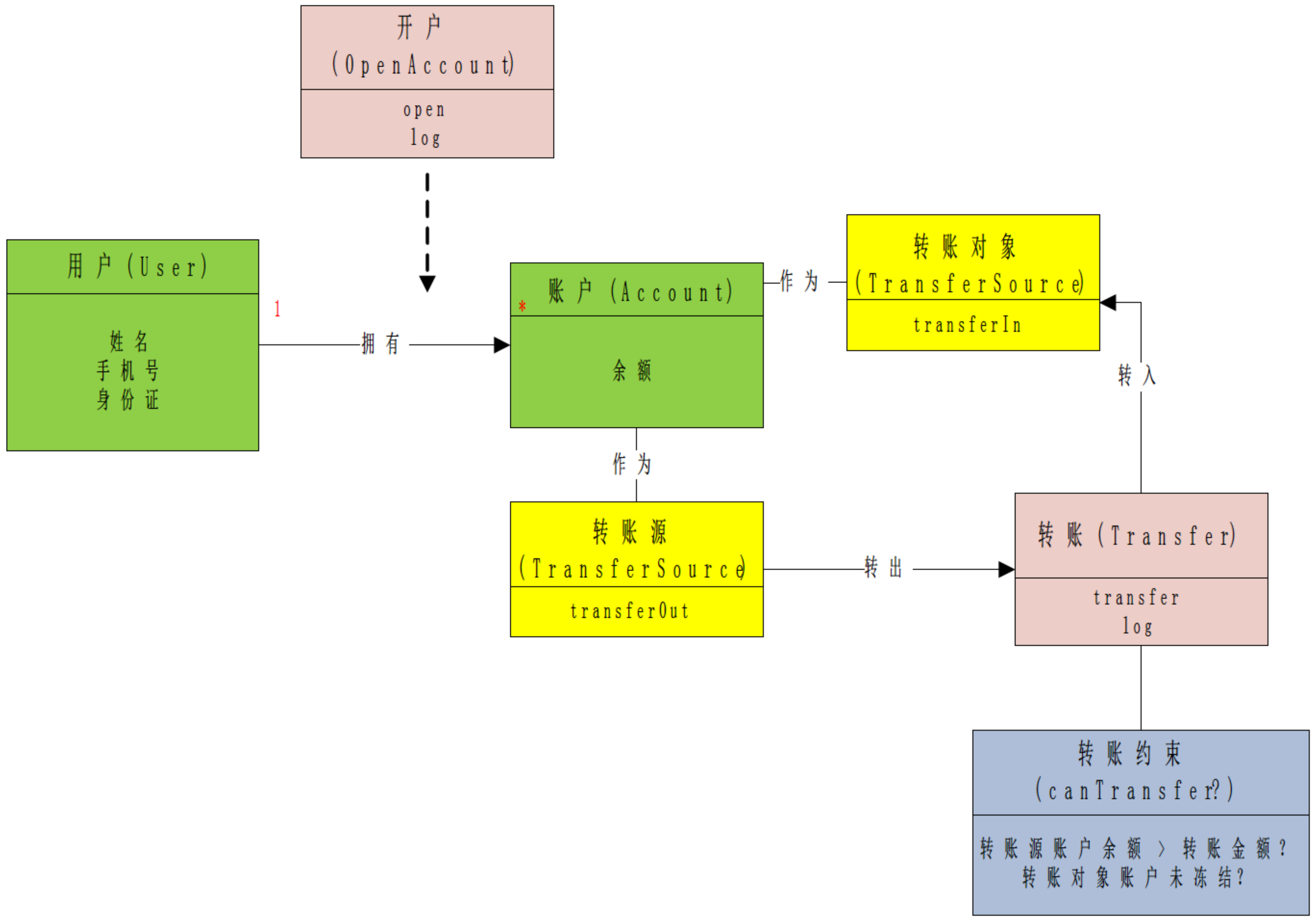
将来的程序员可能只剩一件事

DDD

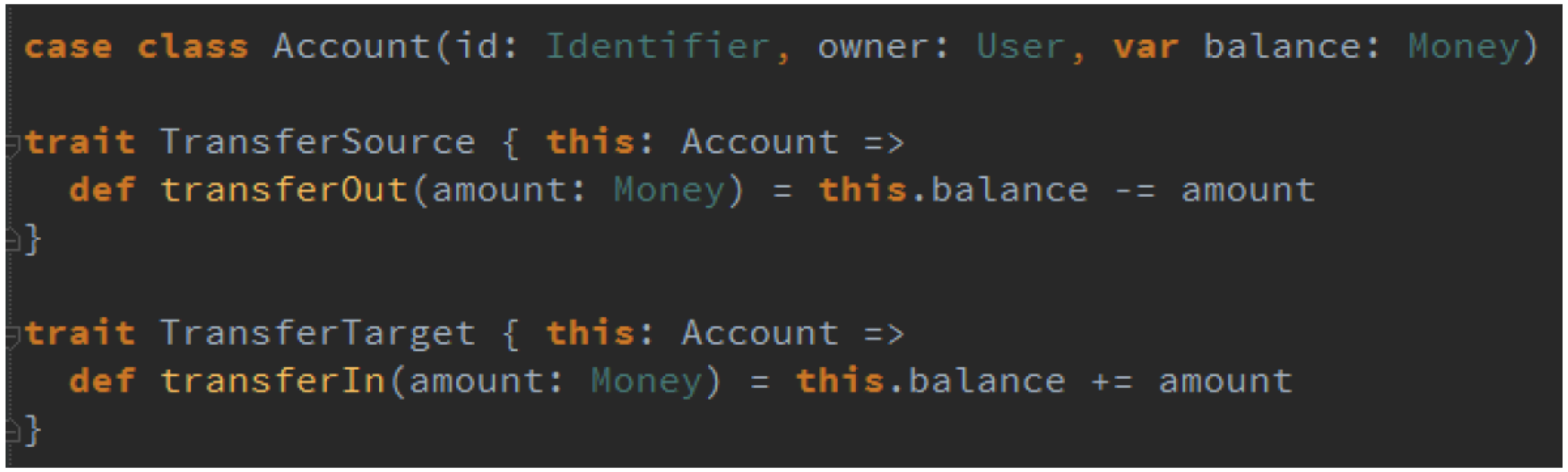
DDD的核心是统一语言



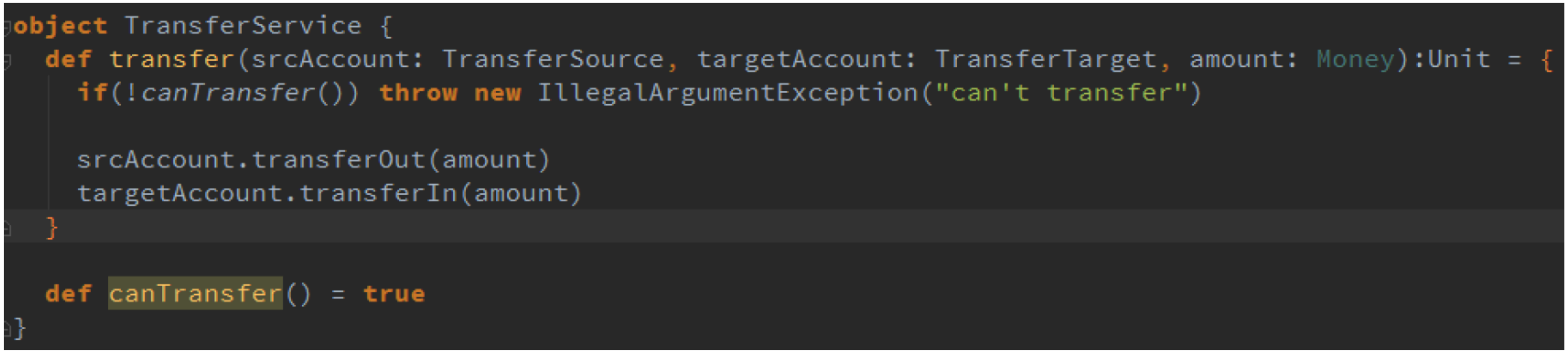
领域建模是不容易的


开户
(OpenAccount)
open
log
用户(User)
姓名
手机号
身份证
1
拥有
*
账户(Account)
余额
作为
转账对象
(TransferSource)
transferIn
转入
作为
转账源
(TransferSource)
transferOut
转出
转账(Transfer)
transfer
log
转账约束
(canTransfer?)
转账源账户余额 > 转账金额?
转账对象账户未冻结?

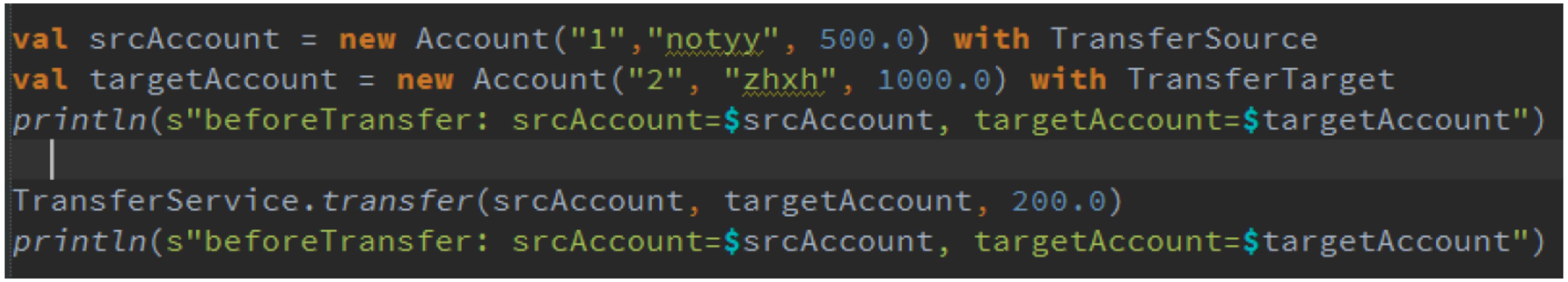
更难的是保持模型和实现的一致



```scala
case class Account(id: Identifier, owner: User, var balance: Money)

trait TransferSource { this: Account =>
  def transferOut(amount: Money) = this.balance -= amount
}

trait TransferTarget { this: Account =>
  def transferIn(amount: Money) = this.balance += amount
}
```



```scala
object TransferService {
  def transfer(srcAccount: TransferSource, targetAccount: TransferTarget, amount: Money):Unit = {
    if(!canTransfer()) throw new IllegalArgumentException("can't transfer")

    srcAccount.transferOut(amount)
    targetAccount.transferIn(amount)
  }

  def canTransfer() = true
}
```



```scala
val srcAccount = new Account("1","notyy", 500.0) with TransferSource
val targetAccount = new Account("2", "zhxh", 1000.0) with TransferTarget
println(s"beforeTransfer: srcAccount=$srcAccount, targetAccount=$targetAccount")

TransferService.transfer(srcAccount, targetAccount, 200.0)
println(s"beforeTransfer: srcAccount=$srcAccount, targetAccount=$targetAccount")
```

函数式编程的核心是?

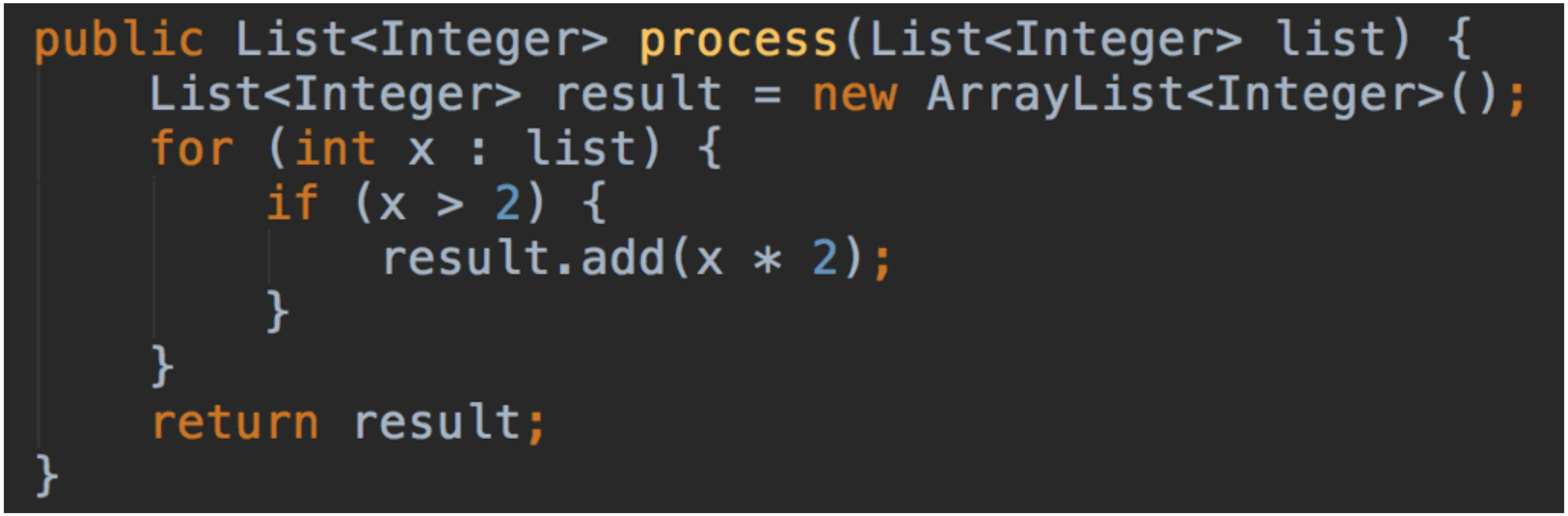
函数是一等公民

Immutability

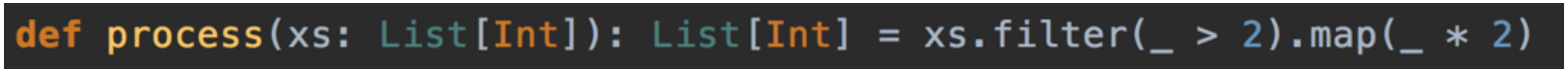
以及

更强的类型系统（大魔头的私货）

让我们描述我们要什么

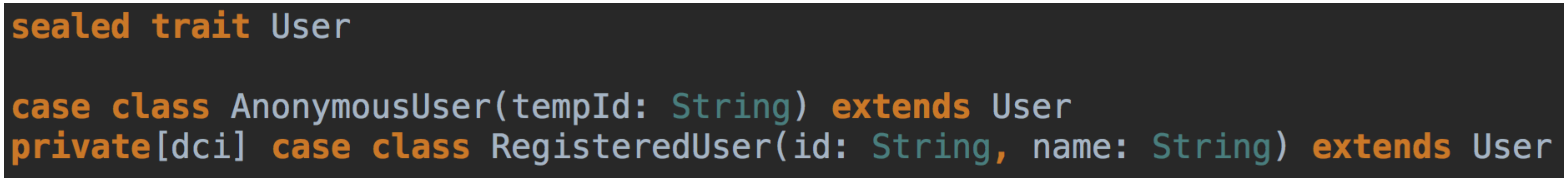


```java
public List<Integer> process(List<Integer> list) {
    List<Integer> result = new ArrayList<Integer>();
    for (int x : list) {
        if (x > 2) {
            result.add(x * 2);
        }
    }
    return result;
}
```

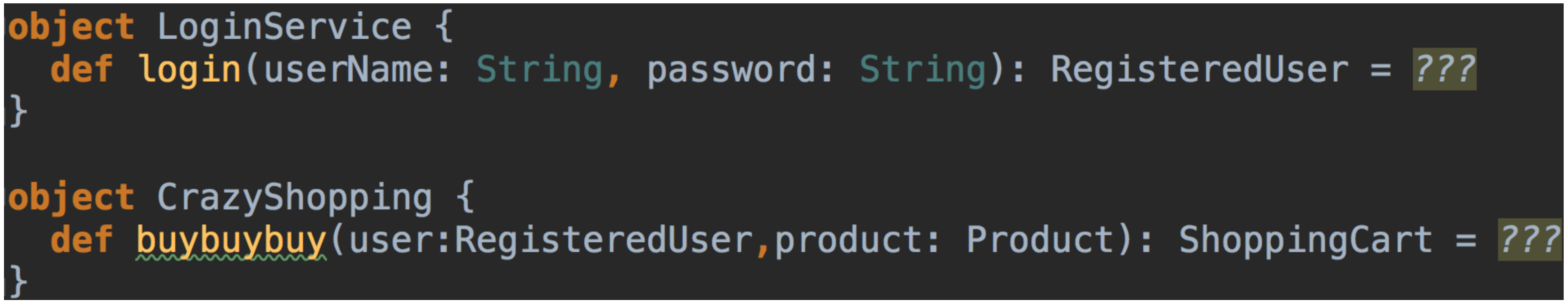


```scala
def process(xs: List[Int]): List[Int] = xs.filter(_ > 2).map(_ * 2)
```

让我们表达业务约束



```scala
sealed trait User

case class AnonymousUser(tempId: String) extends User
private[dci] case class RegisteredUser(id: String, name: String) extends User
```



```scala
object LoginService {
  def login(userName: String, password: String): RegisteredUser = ???
}

object CrazyShopping {
  def buybuybuy(user:RegisteredUser,product: Product): ShoppingCart = ???
}
```

让业务概念有类型可归

```scala
def register(name: String, password: String, email: String, address: String):RegisteredUser
```

```scala
case class UserName(value: String) extends AnyVal
case class Password(value: String) extends AnyVal
case class Email(value: String) extends AnyVal
case class Address(value: String) extends AnyVal
```

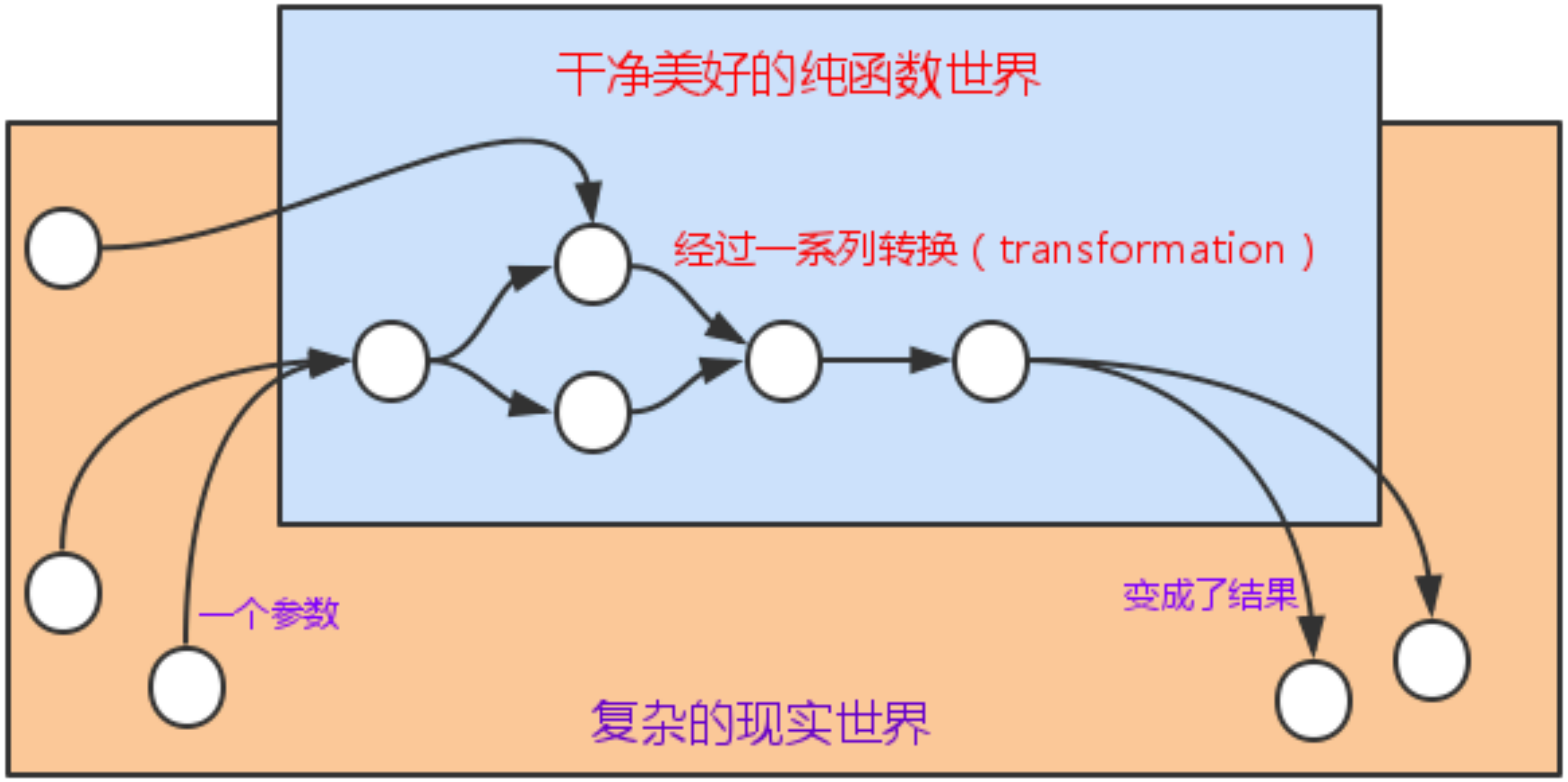
```scala
def register: (UserName, Password, Email, Address) => RegisteredUser
```

我们也可以很纯

```
def register: (UserInfo, Set[RegisteredUser]) => (RegisteredUser, Set[RegisteredUser])
```

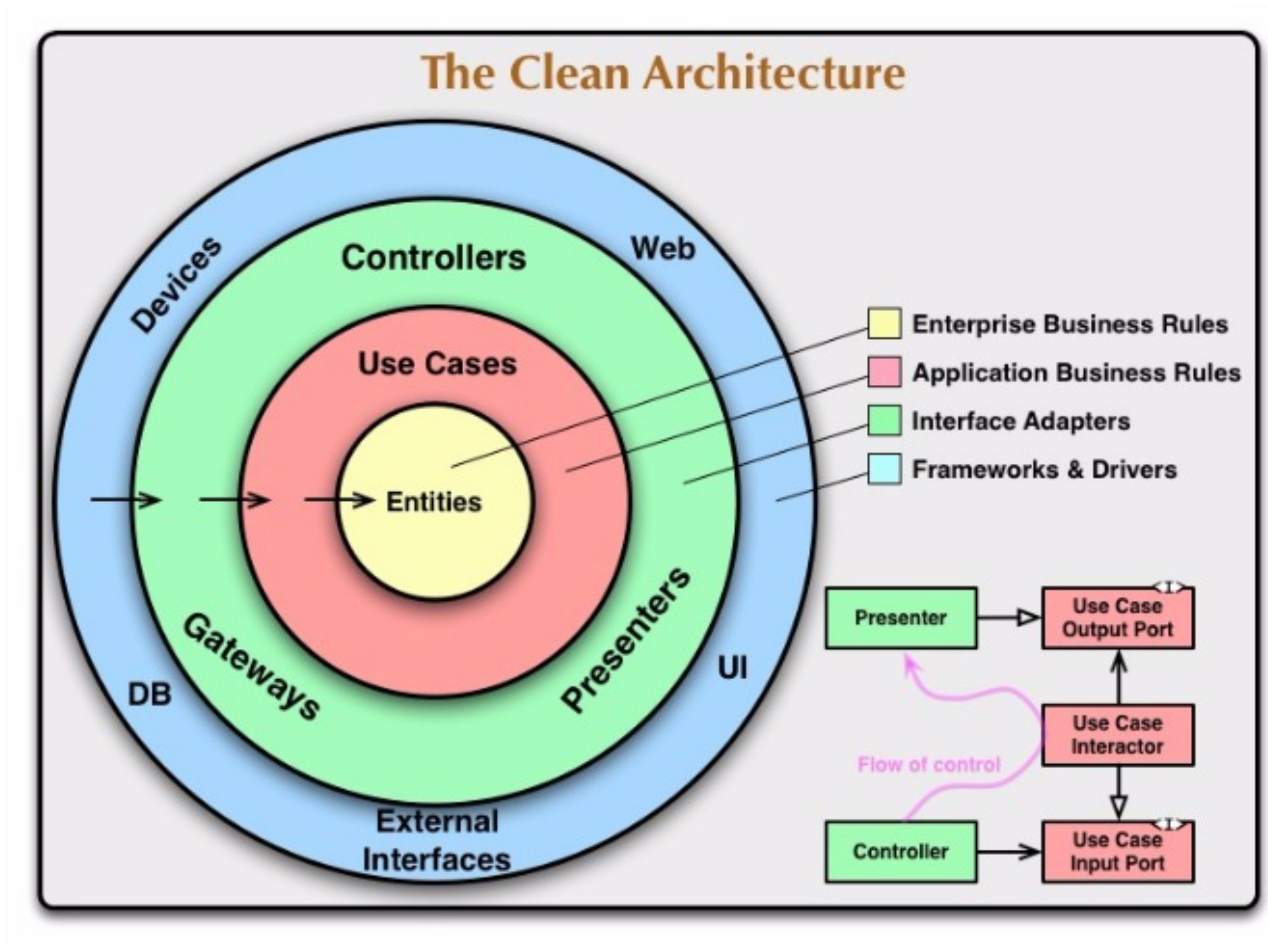
```
update : Msg -> Model -> Model
update msg model =
  case msg of
    Increment ->
      model + 1

    Decrement ->
      model - 1
```

I HAVE A DREAM

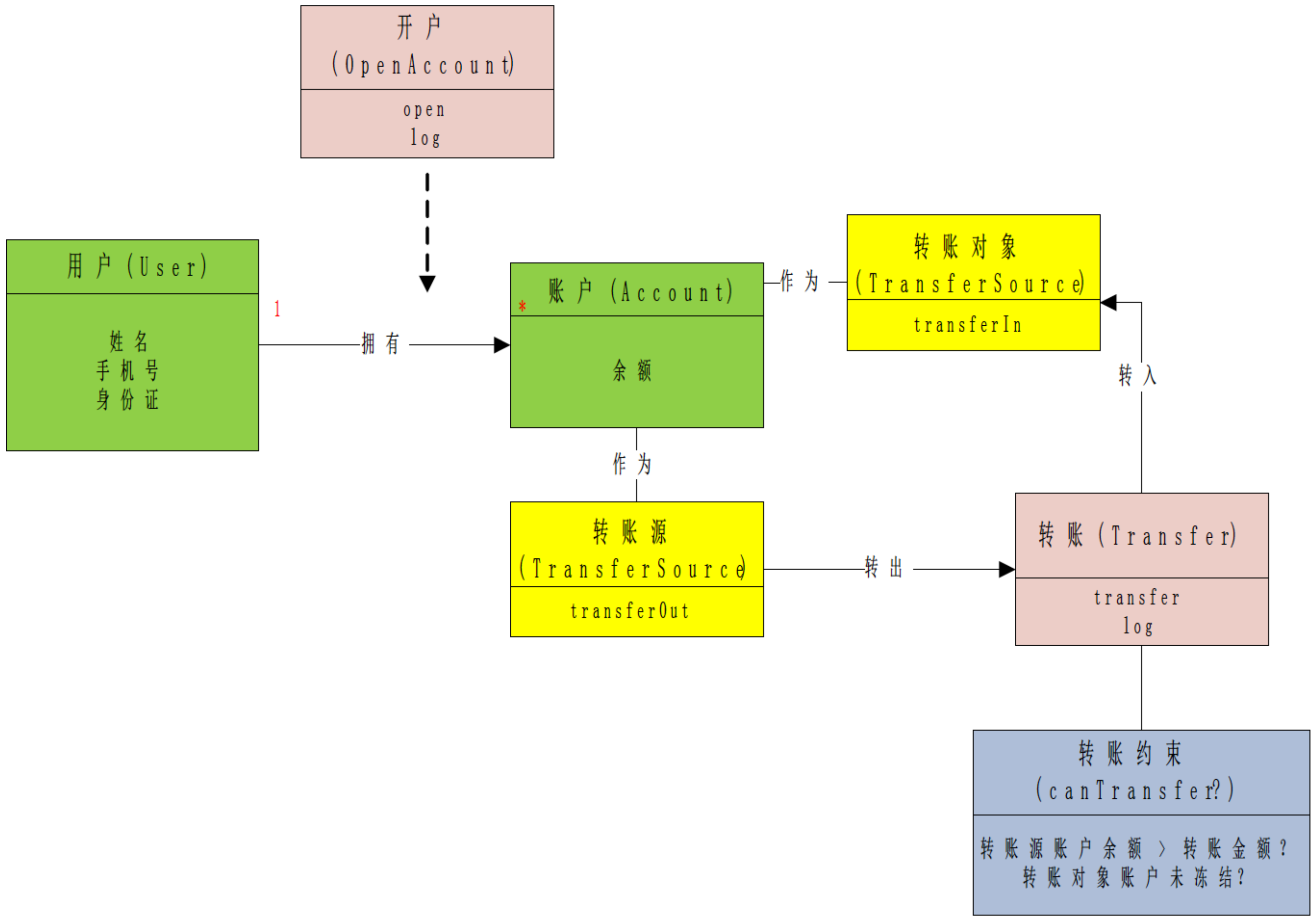


by：大魔头

领域是稳定的，实现方式要与时俱进



最后别忘了模型


开户
(OpenAccount)
open
log
用户(User)
姓名
手机号
身份证
1
拥有
*
账户(Account)
余额
作为
转账对象
(TransferSource)
transferIn
转入
作为
转账源
(TransferSource)
transferOut
转出
转账(Transfer)
transfer
log
转账约束
(canTransfer?)
转账源账户余额 > 转账金额？
转账对象账户未冻结？

DDD是一种纪律