on variance

Jed Wesley-Smith
@jedws

101

- variance is about substitution, or sub-typing
- a thing that produces **Super** *may be replaced by* a thing that produces **Sub**
- a thing that takes **Sub** *may be replaced by* a thing that takes **Super**

101

- if I need a source of Biscuits,
  a packet of Tim Tams will do…
- if I need something to eat Biscuits?

Brian Beckman
@lorentzframe

Following

@jedws @pchiusano @runarorama my slogan is "rat -> real <: int -> complex": a "rat->real" can be applied where you'd take an "int->complex"

Reply Retweet Favorite More

1:22 AM - 25 May 13

java

- has sub-typing
- has type parameters
- has **usage-site** variance

java

```
public interface Function<A, B> {
  B apply(A input);
}
```

```
public class Option<A> {
  public <B> Option<B> map(
    final Function<? super A, ? extends B> f) {…}

  public <B> Option<B> flatMap(
    Function<? super A,
      ? extends Option<? extends B>> f) {…}
}
```

scala

- has sub-typing
- has type parameters
- has **declaration-site** variance

```
List[A]

Function1[T1, R]

Kleisli[M[_], A, B]
```

```
List[+A]

Function1[-T1, +R]

Kleisli[M[+_], -A, +B]
```

```
class GParent
class Parent extends GParent
class Child extends Parent

class Box[+A] // covaraint box
def foo(x : Box[Parent]) : Box[Parent] = identity(x)

foo(new Box[Child])     // success
foo(new Box[GParent])   // type error

class Box2[-A] // contravaraint box
def bar(x : Box2[Parent]) : Box2[Parent] = identity(x)

bar(new Box2[Child])    // type error
bar(new Box2[GParent])  // success
```

so

for a **covariant box**:
a box of sub-types
is a sub-type of
a box of super-types

for a **contravariant box:**
a box of super-types
is a sub-type of
a box of sub-types

covariant

```
trait Box[+A] {
  def get: A
  def take(a: A)
}
```

```
[error] …: covariant type A occurs in contravariant
           position in type A of value a
[error]   def take(a: A)
[error]                ^
```

contravariant

```
trait Box[-A] {
  def get: A
  def take(a: A)
}
```

```
[error] …: contravariant type A occurs in covariant
            position in type => A of method get
[error]   def get: A
[error]       ^
```

functors

```
trait Functor[F[_]]  {
  def map[A, B](a: F[A])(f: A => B): F[B]
}
```

- aka: Covariant Functor!

- wait… what about?

```
trait Contravariant[F[_]]  {
  def contramap[A, B](r: F[A])(f: B => A): F[B]
}
```

function functors

```
type FunctionOf[T] = { type l[a] = Function1[T, a] }

implicit def FunctionFunctor[T] =
  new Functor[FunctionOf[T]#l] {
    def map[A, B](a: T => A)(f: A => B): T => B =
      f compose a
  }

type FunctionTo[T] = { type l[a] = Function1[a, T] }

implicit def FunctionContra[T] =
  new Contra[FunctionTo[T]#l] {
    def contramap[A, B](a: A => T)(f: B => A): B => T =
      a compose f
  }
```

note

- a thing that produces **Super** *may be replaced by* a thing that produces **Sub**
- a thing that takes **Sub** *may be replaced by* a thing that takes **Super**
- **in other words:**
  there is a trivial function Sub => Super

functors > variance

- variance annotations only work for sub-type relationships
- sub-type is implicitly a function `Sub => Super`
- functors generalise to all functions

thanks

ref

Michael Peyton Jones: **Covariance and Contravariance in Scala**
http://termsandtruthconditions.herokuapp.com/blog/2012/12/29/covariance-and-contravariance-in-scala/

Greg Meredith: **Of Monads and Games**
http://biosimilarity.blogspot.com.au/2011/05/of-monads-and-games.html