Intersection Types

Edit this page on GitHub

Used on types, the `&` operator creates an intersection type.

Type Checking

The type `S & T` represents values that are of the type `S` and `T` at the same time.

```
trait Resettable:
  def reset(): Unit

trait Growable[T]:
  def add(t: T): Unit

def f(x: Resettable & Growable[String]) =
  x.reset()
  x.add("first")
```

The parameter `x` is required to be *both* a `Resettable` and a `Growable[String]`.

The members of an intersection type `A & B` are all the members of `A` and all the members of `B`. For instance `Resettable & Growable[String]` has member methods `reset` and `add`.

`&` is *commutative*: `A & B` is the same type as `B & A`.

If a member appears in both `A` and `B`, its type in `A & B` is the intersection of its type in `A` and its type in `B`. For instance, assume the definitions:

```
trait A:
  def children: List[A]

trait B:
  def children: List[B]
```

```
val x: A & B = new C
val ys: List[A & B] = x.children
```

The type of `children` in `A & B` is the intersection of `children`'s type in `A` and its type in `B`, which is `List[A] & List[B]`. This can be further simplified to `List[A & B]` because `List` is covariant.

One might wonder how the compiler could come up with a definition for `children` of type `List[A & B]` since what is given are `children` definitions of type `List[A]` and `List[B]`. The answer is the compiler does not need to. `A & B` is just a type that represents a set of requirements for values of the type. At the point where a value is *constructed*, one must make sure that all inherited members are correctly defined. So if one defines a class `C` that inherits `A` and `B`, one needs to give at that point a definition of a `children` method with the required type.

```
class C extends A, B:
  def children: List[A & B] = ???
```

More details

‹ New Ty... Interse... ›

Contributors to this page

duanebester smarter liufengyun odersky ScalaWilliam pikinier20 michelou BarkingBad julienrf robstoll DaniRey sideeffffect

Scaladoc Copyright (c) 2002-2022, LAMP/EPFL