Kind Polymorphism

Edit this page on GitHub

Normally type parameters in Scala are partitioned into *kinds*. First-level types are types of values. Higher-kinded types are type constructors such as `List` or `Map`. The kind of a type is indicated by the top type of which it is a subtype. Normal types are subtypes of `Any`, covariant single argument type constructors such as `List` are subtypes of `[+X] =>> Any`, and the `Map` type constructor is a subtype of `[X, +Y] =>> Any`.

A type can be used only as prescribed by its kind. Subtypes of `Any` cannot be applied to type arguments whereas subtypes of `[X] =>> Any` *must* be applied to a type argument, unless they are passed to type parameters of the same kind.

Sometimes we would like to have type parameters that can have more than one kind, for instance to define an implicit value that works for parameters of any kind. This is now possible through a form of (*subtype*) kind polymorphism. Kind polymorphism relies on the special type `scala.AnyKind` that can be used as an upper bound of a type.

```
def f[T <: AnyKind] = ...
```

The actual type arguments of `f` can then be types of arbitrary kinds. So the following would all be legal:

```
f[Int]
f[List]
f[Map]
f[[X] =>> String]
```

We call type parameters and abstract types with an `AnyKind` upper bound *any-kinded types*. Since the actual kind of an any-kinded type is unknown, its usage must be heavily restricted: An any-kinded type can be neither the type of a value, nor can it be instantiated with type parameters. So about the only thing one can do with an any-kinded type is to pass it to another any-kinded type argument. Nevertheless, this is enough to achieve some interesting generalizations that work across kinds, typically through advanced uses of implicits.

(todo: insert good concise example)

Some technical details: `AnyKind` is a synthesized class just like `Any`, but without any members. It extends no other class. It is declared `abstract` and `final`, so it can be neither instantiated nor extended.

`AnyKind` plays a special role in Scala's subtype system: It is a supertype of all other types no matter what their kind is. It is also assumed to be kind-compatible with all other types. Furthermore, `AnyKind` is treated as a higher-kinded type (so it cannot be used as a type of values), but at the same time it has no type parameters (so it cannot be instantiated).

Note: This feature is considered experimental but stable and it can be disabled under compiler flag (i.e. `-Yno-kind-polymorphism`).

‹ Param... The M... ›