Wildcard Arguments in Types

Edit this page on GitHub

The syntax of wildcard arguments in types has changed from `_` to `?`. Example:

```
List[?]
Map[? <: AnyRef, ? >: Null]
```

Motivation

We would like to use the underscore syntax `_` to stand for an anonymous type parameter, aligning it with its meaning in value parameter lists. So, just as `f(_)` is a shorthand for the lambda `x => f(x)`, in the future `C[_]` will be a shorthand for the type lambda `[X] =>> C[X]`. This makes higher-kinded types easier to use. It also removes the wart that, used as a type parameter, `F[_]` means `F` is a type constructor whereas used as a type, `F[_]` means it is a wildcard (i.e. existential) type. In the future, `F[_]` will mean the same thing, no matter where it is used.

We pick `?` as a replacement syntax for wildcard types, since it aligns with Java's syntax.

Migration Strategy

The migration to the new scheme is complicated, in particular since the kind projector compiler plugin still uses the reverse convention, with `?` meaning parameter placeholder instead of wildcard. Fortunately, kind projector has added `*` as an alternative syntax for `?`.

A step-by-step migration is made possible with the following measures:

1. In Scala 3.0, both `_` and `?` are legal names for wildcards.
2. In Scala 3.1, `_` is deprecated in favor of `?` as a name for a wildcard. A `-rewrite` option is available to rewrite one to the other.
3. In Scala 3.2, the meaning of `_` changes from wildcard to placeholder for type

parameter.

4. The Scala 3.1 behavior is already available today under the `-source future` setting.

To smooth the transition for codebases that use kind-projector, we adopt the following measures under the command line option `-Ykind-projector`:

1. In Scala 3.0, `*` is available as a type parameter placeholder.
2. In Scala 3.2, `*` is deprecated in favor of `_`. A `-rewrite` option is available to rewrite one to the other.
3. In Scala 3.3, `*` is removed again, and all type parameter placeholders will be expressed with `_`.

These rules make it possible to cross build between Scala 2 using the kind projector plugin and Scala 3.0 - 3.2 using the compiler option `-Ykind-projector`.

There is also a migration path for users that want a one-time transition to syntax with `_` as a type parameter placeholder. With option `-Ykind-projector:underscores` Scala 3 will regard `_` as a type parameter placeholder, leaving `?` as the only syntax for wildcards.

To cross-compile with old Scala 2 sources, while using `_` a placeholder, you must use options `-Xsource:3 -P:kind-projector:underscore-placeholders` together with a recent version of kind-projector (`0.13` and higher) and most recent versions of Scala 2 (`2.13.5` and higher and `2.12.14` and higher)

Rules f... Imports

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