TOUR OF SCALA

TYPE INFERENCE

The Scala compiler can often infer the type of an expression so you don't have to declare it explicitly.

Omitting the type

```
val businessName = "Montreux Jazz Café"
```

The compiler can detect that `businessName` is a String. It works similarly with methods:

```
def squareOf(x: Int) = x * x
```

The compiler can infer that the return type is an `Int`, so no explicit return type is required.

For recursive methods, the compiler is not able to infer a result type. Here is a program which will fail the compiler for this reason:

```
def fac(n: Int) = if (n == 0) 1 else n * fac(n - 1)
```

It is also not compulsory to specify type parameters when polymorphic methods are called or generic classes are instantiated. The Scala compiler will infer such missing type parameters from the context and from the types of the actual method/constructor parameters.

Here are two examples:

```
case class MyPair[A, B](x: A, y: B)
val p = MyPair(1, "scala") // type: MyPair[Int, String]

def id[T](x: T) = x
val q = id(1)              // type: Int
```

The compiler uses the types of the arguments of `MyPair` to figure out what type `A` and `B` are. Likewise for the type of `x`.

Parameters

The compiler never infers method parameter types. However, in certain cases, it can infer anonymous function parameter types when the function is passed as argument.

```
Seq(1, 3, 4).map(x => x * 2)  // List(2, 6, 8)
```

The parameter for map is `f: A => B`. Because we put integers in the `Seq`, the compiler knows that `A` is `Int` (i.e. that `x` is an integer). Therefore, the compiler can infer from `x * 2` that `B` is type `Int`.

When *not* to rely on type inference

It is generally considered more readable to declare the type of members exposed in a public API. Therefore, we recommend that you make the type explicit for any APIs that will be exposed to users of your code.

Also, type inference can sometimes infer a too-specific type. Suppose we write:

```
var obj = null
```

We can't then go on and make this reassignment:

```
obj = new AnyRef
```

It won't compile, because the type inferred for `obj` was `Null`. Since the only value of that type is `null`, it is impossible to assign a different value.

← previous next →