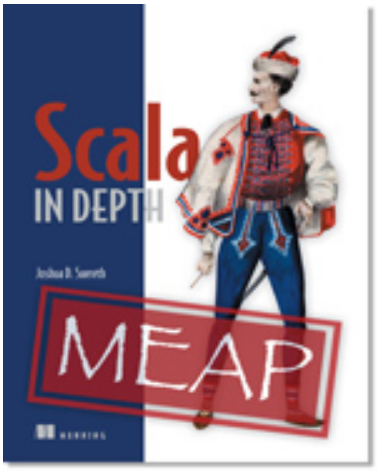
MANNING PUBLICATIONS



Scala in Depth
By Joshua D. Suereth

Scala 2.8 formalized the ability to encode type information into implicit parameters. Manifests were added specifically to handle arrays and generalized to be useful in other situations where the type needs to be available at runtime. In this article from chapter 7 of Scala in Depth, author Joshua Suereth shows how Manifests are useful in Scala to create abstract methods whose implementations diverge based on the types they work on, but the resulting outputs do not.

Using Manifests

Manifests are useful in Scala to create abstract methods whose implementations diverge based on the types they work on, but the resulting outputs do not. A good example of this is any method using generic arrays. Because Scala must use different bytecode instructions depending on the runtime array type, it requires a `ClassManifest` on `the` Array element.

```
scala> def first[A](x : Array[A]) = Array(x(0))
<console>:7: error: could not find implicit value for
evidence parameter of type scala.reflect.ClassManifest[A]
def first[A](x : Array[A]) = Array(x(0))
```

The first method is defined as taking a generic `Array` of type A. It attempts to construct a new array containing just the first element of the old array. However, since we have not captured a manifest, the compiler can't figure out which runtime type the resulting `Array` should have.

```
scala> def first[A : ClassManifest](x : Array[A]) =
     | Array(x(0))
first: [A](x: Array[A])(implicit evidence$1: ClassManifest[A])Array[A]

scala> first(Array(1,2))
res1: Array[Int] = Array(1)
```

Now, the A type parameter also captures the implicit `ClassManifest`. When called with an `Array[Int]`, the compiler constructs a `ClassManifest` for the type `Int`, and this is used to construct a runtime array of the appropriate type.

CLASSMANIFEST AND ARRAYS

The `ClassManifest` class actually directly contains a method to construct new arrays of the type it has captured. This could be used directly instead of delegating to Scala's generic `Array` factory method.

Using manifests requires capturing the manifest when a specific type is known before passing to a generic method. If the type of an array were *lost*, the array could not be passed into the first method.

```
scala> val x : Array[_] = Array(1,2)
x: Array[_] = Array(1, 2)

scala> first(x)
<console>:10: error: could not find implicit value for evidence parameter of type
```

```
ClassManifest[_$first(x)
```

The value `x` is constructed as an Array with existential type. The first method cannot be called with the value `x` because a `ClassManifest` cannot be found for array's type. While this example is contrived, the situation itself occurs when working with Arrays in nested generic code. This is solved by attaching manifests to types all the way back the generic call stack.

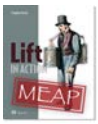
RUNTIME VS. COMPILE TIME

Manifests are captured at compile time and encode the type known at *the time of capture*. This type can then be inspected and used at runtime; however, the manifest can only capture the type available when the Manifest is looked for on the implicit scope.

Summary

Scala 2.8 formalized the ability to encode type information into implicit parameters. This is done through two mechanisms: Manifests and implicit type constraints. A Manifest for a type is generated by the compiler, when needed, with all the known information for that type at that time. Manifests were added specifically to handle arrays and generalized to be useful in other situations where the type needs to be available at runtime.

Here are some other Manning titles you might be interested in:



Lift in Action
Timothy Perrett

Scala in Action
Nilanjan Raychaudhuri

DSLs in Action
Debasish Ghosh

Last updated: September 26, 2011