Spark And Scala 2.12

2018/09/15, Scala Meetup in Hangzhou

沈达 [@sadhen]

Is Scala 2.12 Slower?

- Yes.
- No.

https://github.com/sadhen/scala-benchmark-starter

The Richards Benchmark

```
sbt 'project benchmark' 'set scalaVersion := "2.11.12"' \
'jmh:run -f 1 -i 20 -wi 20 -t 1 Richards'

[info] Benchmark      Mode  Cnt     Score    Error  Units
[info] Richards.run  thrpt   20  7464.953 ± 65.303  ops/s

sbt 'project benchmark' 'set scalaVersion := "2.12.6"' \
'jmh:run -f 1 -i 20 -wi 20 -t 1 Richards'

[info] Benchmark      Mode  Cnt     Score    Error  Units
[info] Richards.run  thrpt   20  6623.674 ± 51.901  ops/s

sbt 'project benchmark' 'set scalaVersion := "2.12.6"' \
'set scalacOptions in ThisBuild ++= " \
"Seq("-opt:l:inline", "-opt-inline-from:**")'
'jmh:run -f 1 -i 20 -wi 20 -t 1 Richards'

[info] Benchmark      Mode  Cnt     Score    Error  Units
[info] Richards.run  thrpt   20  7456.257 ± 23.585  ops/s
```

Scala 2.12 Overview

1. GenBCode and optimizer
2. Trait
3. SAM
4. `invokedynamic`

GenBCode and Optimizer

1. Emits code more quickly because it directly generates bytecode from Scala compiler trees
2. `scalacOptions in ThisBuild ++= Seq("-opt:l:inline", "-opt-inline-from:**")`

TRAITS COMPILE TO INTERFACES

Java 8 Default Method for Interface

Lambda Syntax for SAM Types

```
val r: Runnable = () => System.out.println("Hello")
r.run()
```

```
val r: Runnable = new Runnable {
  override def run(): Unit = {
    System.out.println("Hello")
  }
}
r.run()
```

`invokedynamic` (since Java 7)

Customize the linkage between:

- call site
- method implementation

Compared to Reflection:

- Byte-code level
- Simplified

```java
class Father {
    public static void fatherSay() {
        System.out.println("我是你爸爸");
    }

    public void say() {
        fatherSay();
    }
}

class GrandFather extends Father {
    @Override
    public void say() {
        System.out.println("我是你爷爷");
    }
}

class TianjinStyle extends GrandFather {
    @Override
    public void say() {
        // 天津老爷爷想说：我是你爸爸
        // 但是不能直接用Father类里面的方法
    }
}
```

```java
public class TianjinStyle extends GrandFather {

  private CallSite bootstrapDynamic(
    Lookup caller, String name, MethodType type)
    throws IllegalAccessException, NoSuchMethodException
  {
    MethodHandle mh = caller.findStatic(Father.class,
                name,
                type);
    return new ConstantCallSite(mh);
  }

  @Override public void say() {
    try {
      String name= "fatherSay";
      MethodType type= MethodType.methodType(void.class);
      Lookup lookup= MethodHandles.lookup()

      CallSite say= bootstrapDynamic(lookup, name, type);
      say.getTarget().invokeExact();
    } catch (Throwable ignore) {
    }
  }
}
```

`lambda` in Java 8

```
val r: Runnable = () => System.out.println("Hello")
r.run()
```

```
  public static void main(java.lang.String[]);
    Code:
       0: invokedynamic #2,  0              // InvokeDynai
       5: astore_1
       6: aload_1
       7: invokeinterface #3,  1            // InterfaceMe
      12: return

  private static void lambda$main$0();
    Code:
       0: getstatic     #4                  // Field java/
       3: ldc           #5                  // String xx
       5: invokevirtual #6                  // Method java
       8: return
```

`invokedynamic` for `lambda`

```
       0: invokedynamic #2,  0
       // InvokeDynamic #0:run:()Ljava/lang/Runnable;
```

```
   #2 = InvokeDynamic      #0:#23
   // #0:run:()Ljava/lang/Runnable;
   #23 = NameAndType        #35:#36
   // run:()Ljava/lang/Runnable;
```

```
  0: #20 invokestatic LambdaMetafactory.metafactory:
(Ljava/lang/invoke/MethodHandles$Lookup;
Ljava/lang/String;
Ljava/lang/invoke/MethodType;
Ljava/lang/invoke/MethodType;
Ljava/lang/invoke/MethodHandle;
Ljava/lang/invoke/MethodType;)Ljava/lang/invoke/CallSite;

    Method arguments:
      #21 ()V
      #22 invokestatic JavaLambdaDemo.lambda$main$0:()V
      #21 ()V
```

LambdaMetafactory

```java
public static CallSite metafactory(
  MethodHandles.Lookup caller,
  String invokedName,
  MethodType invokedType,

  MethodType samMethodType,
  MethodHandle implMethod,
  MethodType instantiatedMethodType)
```

- `samMethodType` : Signature and return type of method to be implemented by the function object.
- `instantiatedMethodType` : The signature and return type that should be enforced dynamically at invocation time. This may be the same as samMethodType, or may be a specialization of it.

Spark-14220

1. Scala REPL: Cross Compilation
2. Equaliy for WrappedArray
3. The ClosureCleaner

Scala REPL: Cross Compilation

There are breaking changes in Scala REPL.

Branch condition

```
scala.util.Properties.versionString
```

Separated Source Code

```
scala-2.11
scala-2.12
```

Others

https://github.com/ThoughtWorksInc/enableIf.scala

more to come ...

SBT Convention (wrong case)

```
.
├── scala-2.11
│   └── src
│       └── main
│           └── scala
│               └── 11.scala
├── scala-2.12
│   └── src
│       └── main
│           └── scala
│               └── 12.scala
└── src
    └── main
        └── scala
            └── all.scala
```

SBT Convention

```
.
└── src
    └── main
        ├── scala
        │   └── all.scala
        ├── scala-2.11
        │   └── 11.scala
        └── scala-2.12
            └── 12.scala
```

Equality

```
scala> 1 == 1.0
res0: Boolean = true

scala> Seq(1) == Seq(1.0)
res1: Boolean = true

scala> Array(1) == Array(1.0)
res2: Boolean = false

scala> Array(1).toSeq == Array(1.0).toSeq
res3: Boolean = true

scala> Int.box(1) == Double.box(1.0)
res4: Boolean = true

scala> Seq(Int.box(1)) == Seq(Double.box(1.0))
res6: Boolean = true

scala> Array(Int.box(1)).toSeq == Array(Double.box(1.0)).
res7: Boolean = false

scala> Array(Int.box(1)).toSeq.getClass
class scala.collection.mutable.WrappedArray$ofRef
```

Equality: `hashCode` and `equals`

> Equality with sub-classing is error-prone.

https://github.com/dicarlo2/ScalaEquals

The ClosureCleaner: Why

When Scala constructs a closure, it determines which outer variables the closure will use and stores references to them in the closure object. This allows the closure to work properly even when it's called from a different scope than it was created in.

Scala sometimes errs on the side of capturing too many outer variables (see SI-1419). That's harmless in most cases, because the extra captured variables simply don't get used (though this prevents them from getting GC'd). But it poses a problem for Spark, which has to send closures across the network so they can be run on slaves. When a closure contains unnecessary references, it wastes network bandwidth. More importantly, some of the references may point to non-serializable objects, and Spark will fail to serialize the closure.

Scala 2.12 support: SPARK-14540

Any Questions

```
scala> ???
scala.NotImplementedError: an implementation is missing
  at scala.Predef$.$qmark$qmark$qmark(Predef.scala:284)
  ... 28 elided
```

My External Links

1. https://github.com/sadhen/scala-benchmark-starter
2. https://github.com/texmacs/GNUTeXmacs
3. https://github.com/texmacs/TeXmacs.scala