06: Q33 – Q40 Scala Futures and Promises Interview Q&As

Posted on August 25, 2016 by Arulkumaran Kumaraswamipillai

Q33. What is a “Future” object in Scala? Is it an immutable object?

A33. A Future is a **holder object** that holds a value (i.e. success scenario) or an exception (i.e. failure) which may become available at some point. Future represents the result of an asynchronous computation.

Yes. A “Future” can be assigned once. Once a Future object is given a value or an exception, it becomes in effect immutable.

A “Future” object enables **asynchronous** i.e. non-blocking computation and returns a future holding the result of that

computation as shown below.

```
1
2  import scala.concurrent.{Await, ExecutionContext
3  import scala.concurrent.duration._
4  import scala.util.Failure
5  import scala.util.Success
6
7  import ExecutionContext.Implicits.global //requi
8
9  object SimpleFuture extends App {
10
11   val intFuture: Future[Int] = Future {
12     Thread.sleep(2000);
13     25
14   }
15
16   intFuture.onComplete {
17     case Success(v) => {println("After 2 seconds
18     case Failure(e) => e.printStackTrace();
19   }
20
21   println("I am not blocked");
22   Await.result(intFuture, 5 second) // requires
23 }
24
```

Output:

```
1
2 I am not blocked
3 After 2 seconds => 25
4
```

"scala.concurrent.ExecutionContext.Implicits.global" is an implicit "**ExecutionContext**" that gives you a default ThreadPool with "as many threads as the processors on the machine". There are other options like "CachedThreadPool" for short lived tasks, "FixedThreadPool" for long running tasks, and "ForkJoinPool" for dividing a very large computation into smaller tasks.

Q34. How will you supply a fixed thread pool of size 2 instead of using the default global execution context?
A34. The above "Future { … }" code can be expanded as shown below:

```
1
2   val intFuture: Future[Int] = Future.apply({
```

12: Q78 – Q80 Rec
Spring, Hibernate, & N
Testing & Profiling/Sa
Other Interview Q&A f
Free Java Interviev

16 Technical Key Areas

open all | close all
Best Practice (6)
Coding (26)
Concurrency (6)
Design Concepts (7)
Design Patterns (11)
Exception Handling (3
Java Debugging (21)
Judging Experience In
Low Latency (7)
Memory Managemen
Performance (13)
QoS (8)
Scalability (4)
SDLC (6)
Security (13)
Transaction Managen

80+ step by step Java Tutorials

open all | close all
Setting up Tutorial (6)
Tutorial - Diagnosis (2
Akka Tutorial (9)
Core Java Tutorials (2
Hadoop & Spark Tuto
JEE Tutorials (19)
Scala Tutorials (1)

```
3     Thread.sleep(2000);
4     25
5   })(ExecutionContext.global)
6
```

based on the Scala Future object API method “apply”

```
1
2 def apply[T](body: ⇒ T)(implicit executor: Execut
3
```

The above code can be rewritten with a fixed thread pool of size 2 as shown below.

```
1
2  import scala.concurrent.Await
3  import scala.concurrent.ExecutionContext
4  import scala.concurrent.Future
5  import scala.concurrent.duration.DurationInt
6  import scala.util.Failure
7  import scala.util.Success
8  import java.util.concurrent.Executors
9  import java.util.concurrent.TimeUnit
10
11 object SimpleFuture extends App {
12
13   val pool = Executors.newFixedThreadPool(2)
14   implicit val ec = ExecutionContext.fromExecuto
15
16   val intFuture: Future[Int] = Future.apply({
17     Thread.sleep(2000);
18     25
19   })(ec)
20
21   intFuture.onComplete {
22     case Success(v) => {println("After 2 seconds
23     case Failure(e) => e.printStackTrace();
24   }
25
26   println("I am not blocked");
27   Await.result(intFuture, 5 second) // requires
28
29   //terminate the fixed threadpool
30   pool.awaitTermination(2, TimeUnit.SECONDS)
31   pool.shutdown()
32 }
33
```

The “Future.apply({})()” can be written as shown below.

```
1
2  implicit val ec = ExecutionContext.fromExecutor(
3
4  val intFuture: Future[Int] = Future {
5     Thread.sleep(2000);
```

Spring & HIbernate Tu
Tools Tutorials (19)
Other Tutorials (45)

100+ Java pre-interview coding tests

open all | close all
Can you write code? (
♦ Complete the given
Converting from A to I
Designing your classe
Java Data Structures
Passing the unit tests
What is wrong with th
Writing Code Home A
Written Test Core Jav
Written Test JEE (1)

How good are your .....?

open all | close all
Career Making Know-
Job Hunting & Resum

```
6      25
7    }(ec)
8
```

Q35. Are Await.ready and Await.result blocking calls?
A35. Yes. These are blocking calls.

In the previous example we used the blocking call "**Await.result**" to wait for 5 seconds. In the following example, let's use the "**Await.ready**" blocking call.

```
1
2  import scala.concurrent.{ Await, ExecutionContex
3  import scala.concurrent.duration._
4  import scala.util.Failure
5  import scala.util.Success
6
7  import ExecutionContext.Implicits.global
8  import scala.util.Try
9
10 object SimpleFuture extends App {
11
12   val intFuture: Future[Int] = Future {
13     Thread.sleep(2000);
14     25
15   }
16
17   val result: Try[Int] = Await.ready(intFuture,
18
19   println("I wast blocked for 3 seconds");
20
21   val resultEither: Either[Throwable, Int] = res
22     case Success(v) => Right(v)
23     case Failure(e) => Left(e)
24   }
25
26   resultEither match {
27     case Right(v) => println(v)
28     case Left(e) => e.printStackTrace()
29   }
30
31 }
32
```

Output:

```
1
2 I wast blocked for 3 seconds
3 25
4
```

Q36. Can we use functional combinators like map, flatmap, etc with a Future object?

A36. Yes. Here is an example

```
1
2  import scala.concurrent.{Await, ExecutionContext
3  import scala.concurrent.duration._
4  import scala.util.Failure
5  import scala.util.Success
6
7  import ExecutionContext.Implicits.global
8
9  object SimpleFuture extends App {
10
11   val intFuture: Future[Int] = Future {
12     Thread.sleep(2000);
13     25
14   }
15
16   val formattedFuture: Future[String] = intFutur
17
18   formattedFuture.onComplete {
19     case Success(sv) => {println("After 2 second
20     case Failure(e) => e.printStackTrace();
21   }
22
23   println("I am not blocked");
24   Await.result(formattedFuture, 5 second) //time
25                                           //impo
26 }
27
```

Output

```
1
2 I am not blocked
3 After 2 seconds => result is 25
4
```

Q. How does “5 second” work in the Await.result()?
A. The import “scala.concurrent.duration._” imports “DurationInt” into your scope, and the compiler rewrites “5 second” as

```
1
2 new DurationInt(5).second
3
```

or as shown below since “.” is optional.

```
1
2 DurationInt(5) second
3
```

Q37. Can we use a “for-comprehension” with a Future object?

A37. Yes. Here is an example

```
1
2  import scala.concurrent.{Await, ExecutionContext
3  import scala.concurrent.duration._
4  import scala.util.Failure
5  import scala.util.Success
6
7  import ExecutionContext.Implicits.global
8
9  object SimpleFuture extends App {
10
11   val intFuture: Future[Int] = Future {
12     Thread.sleep(2000);
13     25
14   }
15
16   val formattedFuture: Future[String] = for {
17     result  <- intFuture
18   } yield  {"result is " + result}
19
20
21   formattedFuture.onSuccess {
22     case (sv) => println("After 2 seconds => " +
23   }
24
25   println("I am not blocked");
26   Await.result(formattedFuture, 5 second) //time
27                                           //impo
28 }
29
```

```
1
2 I am not blocked
3 After 2 seconds => result is 25
4
```

Q38. How do you go about chaining Future objects in Scala?

A38. Here is an example using “for-comprehension” where “sumFuture” and “multiplyFuture” kicked off parallel, and then in the “for-comprehension” wait for the results to be available to sum them up.

```
1
2  import scala.concurrent.{ Await, ExecutionContex
3  import scala.concurrent.duration._
4  import scala.util.Failure
5  import scala.util.Success
6
7  import ExecutionContext.Implicits.global
8
9  object SimpleFuture extends App {
```

```
10
11    val x = 23;
12    val y = 12;
13
14    val a = 3;
15    val b = 2;
16
17    val sumFuture: Future[Int] = Future {
18      Thread.sleep(3000); // 3 seconds
19      x + y
20    }
21
22    val multiplyFuture: Future[Int] = Future {
23      Thread.sleep(4000); // 4 seconds
24      a * b
25    }
26
27    println("I am not blocked");
28
29    //(x+y)+(a * b)  = 35 + 6 = 41
30    //chain futures
31    val resultFuture: Future[Int] = for {
32      sumResult <- sumFuture
33      multiplyResult <- multiplyFuture
34    } yield (sumResult + multiplyResult)
35
36    val result = Await.result(resultFuture, 25 sec
37    println("After 4 seconds ....");
38    println("result = " + result); //41
39
40 }
41
```

Output:

```
1
2 I am not blocked
3 After 4 seconds ....
4 result = 41
5
```

Q39. Can you chain Future objects using the functional combinators like map, flatMap, etc?

A39. Yes. A Future gives you a simple way to run an algorithm concurrently. The “sumFuture” and “multiplyFuture” can be run concurrently and then the results can be combined at some point “**flatMap**” and “**map**” **functional combinators**. The “for-comprehension” shown above internally uses the functional combinators.

```
1
2  import scala.concurrent.{ Await, ExecutionContex
3  import scala.concurrent.duration._
4  import scala.util.Failure
5  import scala.util.Success
```

```
6
7  import ExecutionContext.Implicits.global
8
9  object SimpleFuture extends App {
10
11   val x = 23;
12   val y = 12;
13
14   val a = 3;
15   val b = 2;
16
17   val sumFuture: Future[Int] = Future {
18     Thread.sleep(3000); // 3 seconds
19     x + y
20   }
21
22   val multiplyFuture: Future[Int] = Future {
23     Thread.sleep(4000); // 4 seconds
24     a * b
25   }
26
27   println("I am not blocked");
28
29   val resultFuture: Future[Int] = sumFuture.flatl
30     multiplyFuture.map {
31       y => x + y
32     }
33   }
34
35   val result = Await.result(resultFuture, 25 sec
36   println("After 4 seconds ....");
37   println("result = " + result); //41
38
39 }
40
```

Output:

```
1
2  I am not blocked
3  After 4 seconds ....
4  result = 41
5
```

Q40. Can you create a Future object other than using a Future.apply{….} method discussed above?

A40. Yes. Futures can also be created using **Promise**s as shown below.

```
1
2  import java.util.concurrent.Executors
3  import java.util.concurrent.TimeUnit
4
5  import scala.concurrent.ExecutionContext
6  import scala.concurrent.Future
7  import scala.concurrent.Promise
8
```

```
object SimplePromise extends App {

  val pool = Executors.newFixedThreadPool(2)
  implicit val ec = ExecutionContext.fromExecuto

  val p = Promise[Int]
  val f = p.future // get a Future from a Promis

  val producer: Future[Promise[Int]] = Future.ap
    val x = 25;
    p.success{
      println("Sending the value: " + x)
      x
    }
  }(ec)

  val consumer: Future[Unit] = Future.apply {
    f.onSuccess {
      case x =>  println("Received the value: "
    }
  } (ec)

  pool.awaitTermination(5, TimeUnit.SECONDS)
  pool.shutdown()

}


```

Output

```

1
2 Sending the value: 25
3 Received the value: 25
4
```

Whilst a future object is defined as a read-only placeholder created for a result which doesn't yet exist, but will become available in the future, a promise can be thought of as a writable, single-assignment container, which completes a future.

Popular Member Posts

♦ 11 Spring boot interview questions & answers

850 views

♦ Q11-Q23: Top 50+ Core on Java OOP Interview Questions & Answers

769 views

001A: ♦ 7+ Java integration styles & patterns interview questions & answers

399 views

18 Java scenarios based interview Questions and Answers

387 views

♦ 7 Java debugging interview questions & answers

308 views

01b: ♦ 13 Spring basics Q8 – Q13 interview questions & answers

305 views

01: ♦ 15 Ice breaker questions asked 90% of the time in Java job interviews with hints

297 views

♦ 10 ERD (Entity-Relationship Diagrams) Interview Questions and Answers

294 views

♦ Q24-Q36: Top 50+ Core on Java classes, interfaces and generics interview questions & answers

246 views

001B: ♦ Java architecture & design concepts interview questions & answers

204 views

Bio | Latest Posts

Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via Amazon.com in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site.**945+** paid members. join my LinkedIn Group. **Reviews**

About Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job

interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via Amazon.com in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site.**945+** paid members. join my LinkedIn Group. **Reviews**

‹ SSH Login without password so that tasks can be automated via shell scripts

06: Spark Streaming with Flume Avro Sink Tutorial ›

Posted in member-paid, Scala Interview Q&As

Empowers you to open more doors, and fast-track

Technical Know Hows

☀ Java generics in no time ☀ Top 6 tips to transforming your thinking from OOP to FP ☀ How does a HashMap internally work? What is a hashing function?
☀ 10+ Java String class interview Q&As ☀ Java auto un/boxing benefits & caveats ☀ Top 11 slacknesses that can come back and bite you as an experienced Java developer or architect

Non-Technical Know Hows

☀ 6 Aspects that can motivate you to fast-track your career & go places ☀ Are you reinventing yourself as a Java developer? ☀ 8 tips to safeguard your Java career against offshoring ☀ My top 5 career mistakes

Prepare to succeed

☀ Turn readers of your Java CV go from “Blah blah” to “Wow”? ☀ How to prepare for Java job interviews? ☀ 16 Technical Key Areas ☀ How to choose from multiple Java job offers?

Select Category ▼

© Disclaimer