Futures with Scala

Markus Jais

What is a future?

C++11

Class future<> represents an outcome of an operation. It can be a return value or an exception but not both.

From: The C++ Standard library, 2nd edition by Nicolai M. Josuttis

Java 7

A Future represents the result of an asynchronous computation.

The result can only be retrieved using method get when the computation has completed, blocking if necessary until it is ready.

From: Java 7 javadocs

Java 7 Futures - not powerful enough

- only limited API

- blocking operations

Java 7 Futures - Antipattern

```
Future<String> future = getFuture();
String result = future.get();
```

Problem: immediately call a blocking operation. This basically makes Futures useless.

Scala (>= 2.10)

A future is a pocket of concurrency that executes independently of the calling thread and ultimately represents the “return value” from that code.

(From: Akka Concurreny by Derek Wyatt)

Scala (>= 2.10)

A future is an abstraction which represents a value which may become available at some point.

(From: SIP-14 - Futures and Promises)

Scala (>= 2.10)

A promise can be used to successfully complete a future with a value (by “completing” the promise) using the success/failure method.

(From: SIP-14 - Futures and Promises)

Future vs. Promise

A *future* is a **READ** handle.

A *promise* is a **WRITE** handle.

Scala Future vs. Java Future

- Non-blocking composition

- Cannot be canceled

Java 8 - CompletableFuture

A Future that may be explicitly completed (setting its value and status), and may include dependent functions and actions that trigger upon its completion.

From: Java 8 javadocs

A Future can have 3 states

Pending

Success | Failure

A simple Example

Code Demo

WTF is an ExecutionContext?

scala.concurrent.ExecutionContext.Implicits.global

Or use your own ExecuterService

Rule number one

Don't block!

Futures are functional

map / flatMap / filter / for comprehensions

More about composing Futures

traverse / sequence / fold

More useful stuff

find / collectFirst

When things go wrong

fallbackTo / recover

Ordering

andThen

Futures: Best practices

Again:

Don't block

Futures: Best practices

Avoid callbacks!

better:

for comprehensions

Futures: Best practices

Know your API

Futures: Best practices

Test performance, thread usage, etc

Futures: Best practices

Combine with Akka actors

(upcoming talk)

Futures: Best practices

Don't close over mutable state!

Further Information

www.akka.io

http://docs.scala-lang.org/overviews/core/futures.html

Videos by Viktor Klang

(google search)

Further Information

Akka Concurrency

Building reliable software in a multi-core world

by Derek Wyatt

```
pizza.foldLeft(emptyStomach)(_ + _)
```