Futures in Scala and Java 8

Futures: Handling Asynchronous Computation

- In Asynchronous computation, we have to deal with uncertain timing of operations – rather than “blocking” until an operation completes, it will complete when it naturally does
- These represent “future values” that can handle now – and which will go on to successive stages of processing when the value is available
- In contrast to distributed callbacks, these capture the successive stages of process in a central way

Example: Long-Latency Operations

- Suppose we want to
  - Wait for user input for search term
  - Perform search using search term
  - Harvest contents from URLs resulting from search
  - Perform an intensive sentiment analysis on the results with respect to some sentiment
  - Summarize the sentiment

Scala: Some Future[T] Operators

- isCompleted: Boolean
- value: Option[Try[T]]
  - Not yet completed: None
  - Completed: Holds the Try value (itself indicating success or failure)
- result(timeOutThreshold:Duration): T (blocks for result)
- map[S](f: T=>S): Future[S]
- flatMap[S](f: T=>Future[S]): Future[S]
- filter(T=>Boolean):
- collect[U](PartialFunction[T,U]): Future[U]
- onComplete(v: Try[T]): Unit
- onSuccess[U](PartialFunction[T,U]): Unit
- onFailure[U](PartialFunction[Throwable, U]): Unit

Advantages

- Standard blocking computation: wait when performing things directly
- Can define and hold computations while performing other operations

Recall Example: Long-Latency Operations

- Suppose we want to
  - Wait for user input for search term
  - Perform search using search term
  - Harvest contents from URLs resulting from search
  - Perform an intensive sentiment analysis on the results with respect to some sentiment
  - Summarize the sentiment

Creating a Future with “future” in Scala

- future { code }
- Examples
  - import scala.concurrent._
  - import scala.io._
  - import scala.util._
  - import scala.concurrent.ExecutionContext.Implicits.global
  - val promise = future {Source.fromURL("http://www.usask.ca").mkString }
  - promise.isCompleted
  - val promise2 = future {Source.fromURL("http://www.cnn.com") }.map(src => src.getLines.toArray.flatMap(_.split(raw"[^a-zA-Z]+")).filter(word => word.length > 0).groupBy(word => word).mapValues(groupedWords => groupedWords.size).toSeq.sortWith((pair1, pair2) => pair1._2 > pair2._2))

Building Block Methods: Scala

getSubmittedSearchTerms() : Future[String]

performWebSearch(String searchTerms) : Future[URL]

extractLinks(String) : List[String]

readURLContents(URL: String) : Future[String]

performPageSentimentAnalysis(String) : Future[Double]

def AveragingDouble(): Double

Example Completable Future: Scala (Currently Rough)

```
double sentimentSumCount = this.getSubmittedSearchTerms()
                .flatMap(performWebSearch)
                .map(_.extractLinks)
                .filter(_.contains("foo.com"))
               .map(readURLContents)
                .map(flatMap(performPageSentimentAnalysis))
                . (stream.foldLeft((0.0,0))((p,s) => (p._1 + s, p._2+1)))
```

Future[String]

Future[URL]

Future[List[URL]]

(Parallel) Stream[URL]

(Parallel) Stream[Future[String]]

Stream[Future[Double]]

(Double,Int)