

Exercise 6: Futures and Actors



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Concepts and Technologies for Distributed Systems and Big Data Processing – SS 2016

Solution 1 Using *Future* combinators

The following code reads the content of a file (`IO.read`), parses it as JSON (`JSON.parse`) and computes a result (`someComplexComputation`) based on the JSON dataset if the dataset is not marked to be ignored (`!json.hasProperty(Ignore)`). The respective methods return their result as future for the sake of performance.

```
1 val content = IO.read("filename.json")
2 content.onComplete {
3   case Success(content) =>
4     val json = JSON.parse(content)
5     json.onComplete {
6       case Success(json) =>
7         if (!json.hasProperty(Ignore)) {
8           val result = Future {
9             someComplexComputation(json)
10          }
11
12          result.onComplete {
13            case Success(result) =>
14              println(s"final result: $result")
15            case Failure(_) =>
16              println("could not compute result")
17          }
18        }
19        else
20          println("could not compute result")
21
22        case Failure(_) =>
23          println("could not compute result")
24      }
25
26   case Failure(_) =>
27     println("could not compute result")
28 }
```

Rewrite the computation using Future combinators such as `flatMap`, `map` and `filter` instead of `onComplete` callbacks.

```
1 (IO.read("filename.json")
2   flatMap JSON.parse
3   filter { !_.hasProperty(Ignore) }
4   map someComplexComputation
5   onComplete {
6     case Success(result) =>
7       println(s"final result: $result")
8     case Failure(_) =>
9       println("could not compute result")
10  })
```

Solution 2 Implementing *Future* combinators

Implement the Future map combinator in terms of `async/await`, which creates a new future by applying the function given by mapping to the result of the given future.

```
1 def map[T, U](future: Future[T], mapping: T => U): Future[U] =
2   async {
3     mapping(await(future))
4   }
```

Implement the Future filter combinator in terms of `async/await`, which filters the value of the given future with the given predicate. If the value of the future satisfies the predicate, the result will hold the same value. Otherwise, the computation will fail.

```
1 def filter[T](future: Future[T], predicate: T => Boolean): Future[T] =
2   async {
3     val value = await(future)
4     if (!predicate(value))
5       throw new NoSuchElementException
6     else
7       value
8   }
```