Implementation of FlatMap

Principles of Functional Programming

Implementation of FlatMap

Let's take a closer look at `flatMap`:

```
trait Future[T] {
  def onComplete(callback: Try[T] => Unit) = ...
  def flatMap[S](f: T => Future[S]): Future[S] = ???
}
```

How can we implement `flatMap` in terms of `onComplete`?

Here's a simplified implementation.

In fact, that implementation is almost automatic; all we need to do is *follow the types*.

Implementation of FlatMap

Start the implementation by creating a result future.

```
trait Future[T] { self =>
  def flatMap[S](f: T => Future[S]): Future[S] =
    new Future[S] {
      ...




    }
```

Implementation of FlatMap

We need to provide its `onComplete` method:

```
trait Future[T] { self =>
  def flatMap[S](f: T => Future[S]): Future[S] =
    new Future[S] {
      def onComplete(callback: Try[S] => Unit): Unit =
        ...



    }
```

Implementation of FlatMap

The obvious thing to do is consult the current future via `self.onComplete`:

```
trait Future[T] { self =>
  def flatMap[S](f: T => Future[S]): Future[S] =
    new Future[S] {
      def onComplete(callback: Try[S] => Unit): Unit =
        self onComplete {
          case Success(x) => ...
          case Failure(e) => ...
        }
      }
    }
```

Implementation of FlatMap

If that returns a value `x`, compute `f(x)`, ...

```
trait Future[T] { self =>
  def flatMap[S](f: T => Future[S]): Future[S] =
    new Future[S] {
      def onComplete(callback: Try[S] => Unit): Unit =
        self onComplete {
          case Success(x) => f(x) ...
          case Failure(e) => ...
        }
      }
    }
```

Implementation of FlatMap

If that returns a value `x`, compute `f(x)`, and pass its result to `callback`.

```
trait Future[T] { self =>
  def flatMap[S](f: T => Future[S]): Future[S] =
    new Future[S] {
      def onComplete(callback: Try[S] => Unit): Unit =
        self onComplete {
          case Success(x) => f(x).onComplete(callback)
          case Failure(e) => ...
        }
      }
    }
```

Implementation of FlatMap

In case of failure, pass it along directly to `callback`.

```
trait Future[T] { self =>
  def flatMap[S](f: T => Future[S]): Future[S] =
    new Future[S] {
      def onComplete(callback: Try[S] => Unit): Unit =
        self onComplete {
          case Success(x) => f(x).onComplete(callback)
          case Failure(e) => callback(Failure(e))
        }
      }
    }
```

Implementation of FlatMap

In case of failure, pass it along directly to `callback`.

```
trait Future[T] { self =>
  def flatMap[S](f: T => Future[S]): Future[S] =
    new Future[S] {
      def onComplete(callback: Try[S] => Unit): Unit =
        self onComplete {
          case Success(x) => f(x).onComplete(callback)
          case Failure(e) => callback(Failure(e))
        }
      }
    }
```

The actual implementation is somewhat more involved since it also has to handle thread scheduling.