UDC 004.75

V.V. Spirintsev, A.A.Kushka

FRAMEWORK ON THE BASIS OF THE SCALA PROGRAMMING LANGUAGE FOR CREATION RESTFUL WEB SERVICES

Abstract. Framework on the basis of the Scala programming language for development of the standardized RESTful of web services is offered. The project is aimed at application by other software developers for facilitation of development of web applications created by means of microservices architecture.
Keywords: framework, web services, Scala, REST.

Problem. One of the most important issues of development of web applications is process of interaction of the client with the server. Long time this process was arbitrary for each application. Such situation existed before creation of the first protocols of serialization of data and mechanisms for creation of arbitrary requests. Appearance of standards for communication between the client and the server allowed to solve a problem of creation of different clients for the same server. It is necessary to mark that at the moment there is a row of problems during creation of the standardized and effective server part: a high threshold of an input for implementation of similar systems (connected to absence of complex tools for design and implementation of web services) existence of certain experience in development of web applications (RESTful web services should be brought together from separate components [1]); absence or small quantity of examples of the organization of web services in the application (it is connected to closeness of source codes or even any details of implementation). Therefore researches in the direction of development of a server part on the basis of web services are urgent.

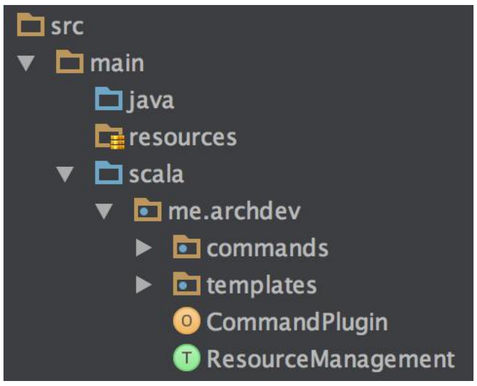
Analysis of researches. Dynamic growth of audience of the Internet network is now watched. For provision of high-quality services to users infrastructure of the modern web applications shall be scaled easily and timely. But in case of increase in the project it becomes more difficult to observe stability of the application, and the speed of writing of a new functionality decreases. The solution of this problem is partition of the application on web services, but it increases quantity of unexpected situations (a problem with a network, unexpected change of the data transfer protocol, etc.) [2,3]. For their preventing it is necessary to use tools which will warn the developer about existence of errors. An effective remedy for this purpose is the choice of infrastructure of JVM and the Scala [4] programming language which has the powerful static analyzer and support as an object-oriented paradigm and the functional programming paradigm.

The aim of article is creation of a framework for rapid development of RESTful of web services on the basis of the Scala programming language which will organize operation between components of system and will solve the following problems: creation of structure of the application; unification of interaction with a web service; organizations of asynchronous operation of a web service; service caching settings; assemblies of the ready application and its delivery to the server.

Major part. For creation of a framework the Scala programming language was selected. From instruments of compilation of a code, available to a JVM platform, the sbt [5] system was selected. As a work environment the computer on the basis of OS X OS with use of the virtual machine Java of version 8 and Scala version 2.11.8 compiler is used. As code editor IntelijIdea 2016.1 is selected.

Creation of structure of a framework

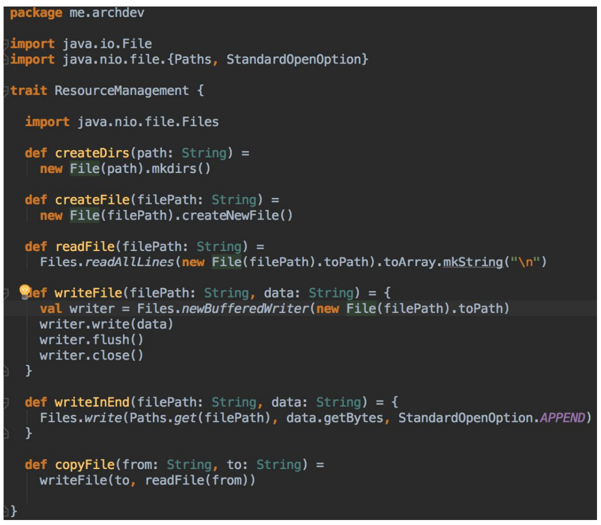
The structure of a framework is given in fig. 1. The src folder contains all adobe codes of a framework in a root of the project. The main folder is responsible for adobe codes of the main application. In the middle of the main folder we create the resources folders for static files and scala for an application code. In a directory we create applications me.archdev packet (contains packets of commands and templates). Commands contains classes for implementation of the main functionality of a framework, templates contains bases for classes which the framework will create during an operating time. The created class CommandPlugin is used as the aggregator of all available commands and the main class of a plug-in. ResourceManagement is a class for interaction with file system of the user of a plug-in.


```
src
▼ main
    java
    resources
  ▼ scala
    ▼ me.archdev
      ▶ commands
      ▶ templates
        CommandPlugin
        ResourceManagement
```
Figure 1 - Structure of framework

We set up the instrument of assembly to have an opportunity to create snapshota for testing. For this purpose in a root of the project we create the build.sbt file and we configure its settings.

Creation of a resource manager

One of the main classes for a framework is the resource manager (it is used for creation and filling of files with a code of the program of the user). This problem is solved thanks to a packet of java.nio which will organize dataful interaction in file system by means of buffers. The class will be developed in a format of a treyt - it is the interface which can have basic implementation (fig. 2).

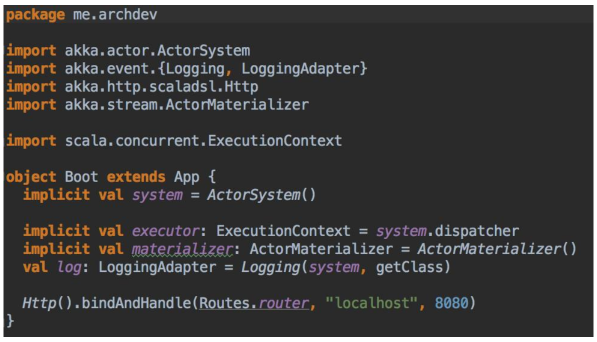


```
package me.archdev

import java.io.File
import java.nio.file.{Paths, StandardOpenOption}

trait ResourceManagement {

  import java.nio.file.Files

  def createDirs(path: String) =
    new File(path).mkdirs()

  def createFile(filePath: String) =
    new File(filePath).createNewFile()

  def readFile(filePath: String) =
    Files.readAllLines(new File(filePath).toPath).toArray.mkString("\n")

  def writeFile(filePath: String, data: String) = {
    val writer = Files.newBufferedWriter(new File(filePath).toPath)
    writer.write(data)
    writer.flush()
    writer.close()
  }

  def writeInEnd(filePath: String, data: String) = {
    Files.write(Paths.get(filePath), data.getBytes, StandardOpenOption.APPEND)
  }

  def copyFile(from: String, to: String) =
    writeFile(to, readFile(from))

}
```

Figure 2 - The Example of a treyt

Creation of classes for start of a Web server

In order that the user a framework had an opportunity right after creation of the project to launch a Web server, we need to create a base class (fig. 3). Further we set up and we launch a Web server. As a Web server the akka-http library which provides to us will be used http-server is written in the Scala language. The selected Web server is asynchronous and multithreaded. Its main configuration is the file with routing of requests, in it the user of a framework shall describe structure of links for access to RESTful of resources. After creation of function for generation of base classes we shall assign a framework command for a call of this function from the console of the instrument of compilation sbt. For this purpose we use TaskKey with the name provision.rezultatom of operation there will be a creation of two classes in a standard packet of the user: one for the description routing, and another for start of a web service.

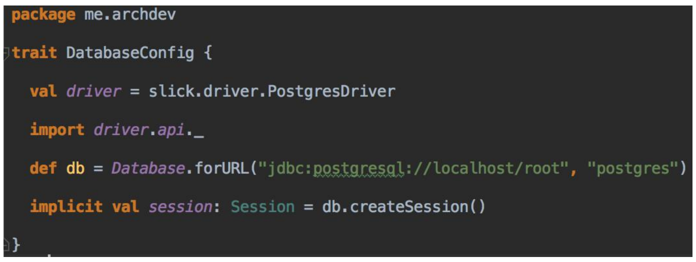


```
package me.archdev

import akka.actor.ActorSystem
import akka.event.{Logging, LoggingAdapter}
import akka.http.scaladsl.Http
import akka.stream.ActorMaterializer

import scala.concurrent.ExecutionContext

object Boot extends App {
  implicit val system = ActorSystem()

  implicit val executor: ExecutionContext = system.dispatcher
  implicit val materializer: ActorMaterializer = ActorMaterializer()
  val log: LoggingAdapter = Logging(system, getClass)

  Http().bindAndHandle(Routes.router, "localhost", 8080)
}
```

Figure 3 - The Basic class

Setup of operation with the database

One of the most important features of RESTful of web services is absence of an internal state. It means that all data shall be stored separately, that is in databases. We use the JDBC driver for access to the database of the user of a framework. To facilitate development of a web service on



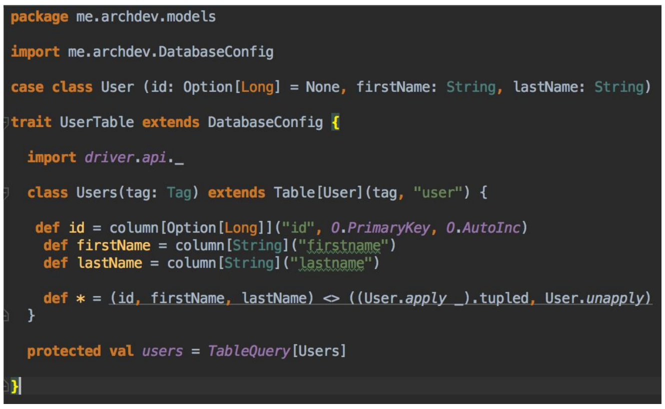
```
package me.archdev

trait DatabaseConfig {

  val driver = slick.driver.PostgresDriver

  import driver.api._

  def db = Database.forURL("jdbc:postgresql://localhost/root", "postgres")

  implicit val session: Session = db.createSession()

}
```

Figure 4 – The Class for attuning of access to the database

the created framework we add one more abstraction layer for generation of a SQL code. The library of the Scala programming language - Slick will be for this purpose used. Feature of this library is that it brings conversion of Scala of a code into SQL in compilation time of the project. Thanks to it the result of operation is always checked by the static analyzer of the Scala language, and the functionality of library doesn't influence the speed of execution of a code as in a case with analogs in the Java language. From a framework we shall provide to the user a method to use already configured Slick library in the code. For this purpose we create трейт DatabaseConfig which will be added to each class of the user where work with the database (fig. 4) will be performed.

Creation of a data model

Basis of RESTful of a web service is the data model with which it will work. For this purpose we form a team which will collect data of the user on necessary models and to generate a class with necessary data for storage of this model in the database (fig. 5).

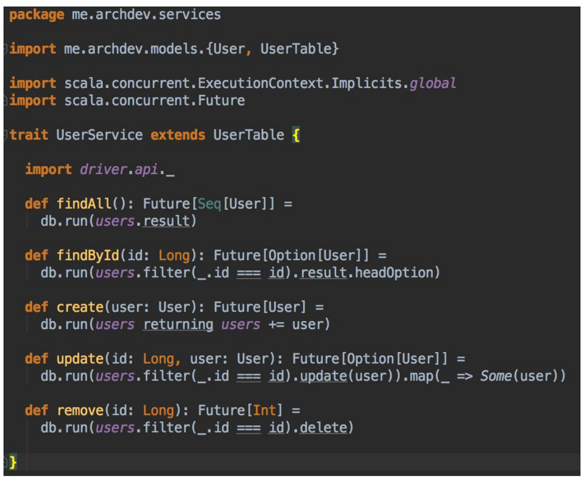


```
package me.archdev.models

import me.archdev.DatabaseConfig

case class User (id: Option[Long] = None, firstName: String, lastName: String)

trait UserTable extends DatabaseConfig {

  import driver.api._

  class Users(tag: Tag) extends Table[User](tag, "user") {

   def id = column[Option[Long]]("id", O.PrimaryKey, O.AutoInc)
    def firstName = column[String]("firstname")
    def lastName = column[String]("lastname")

    def * = (id, firstName, lastName) <> ((User.apply _).tupled, User.unapply)
  }

  protected val users = TableQuery[Users]

}
```

Figure 5 – The Example of a data model

Creation of basic actions of REST service and creation of REST of routing for operation with service

Having model for data storage and basic actions over the database we can generate service which will provide basic actions for REST service. For this purpose we will generate трейт which will be inherited from a treyt with the description of a mapping between the database and model, and we realize operations which will be: to remove all records, to remove separate record on the basis of its identification code, to add entries, to update and delete records. For implementation of these actions we will use Slick library. By means of the basic actions which are already created by a macro with the database, we describe necessary



```
package me.archdev.services

import me.archdev.models.{User, UserTable}

import scala.concurrent.ExecutionContext.Implicits.global
import scala.concurrent.Future

trait UserService extends UserTable {

  import driver.api._

  def findAll(): Future[Seq[User]] =
    db.run(users.result)

  def findById(id: Long): Future[Option[User]] =
    db.run(users.filter(_.id === id).result.headOption)

  def create(user: User): Future[User] =
    db.run(users returning users += user)

  def update(id: Long, user: User): Future[Option[User]] =
    db.run(users.filter(_.id === id).update(user)).map(_ => Some(user))

  def remove(id: Long): Future[Int] =
    db.run(users.filter(_.id === id).delete)

}
```

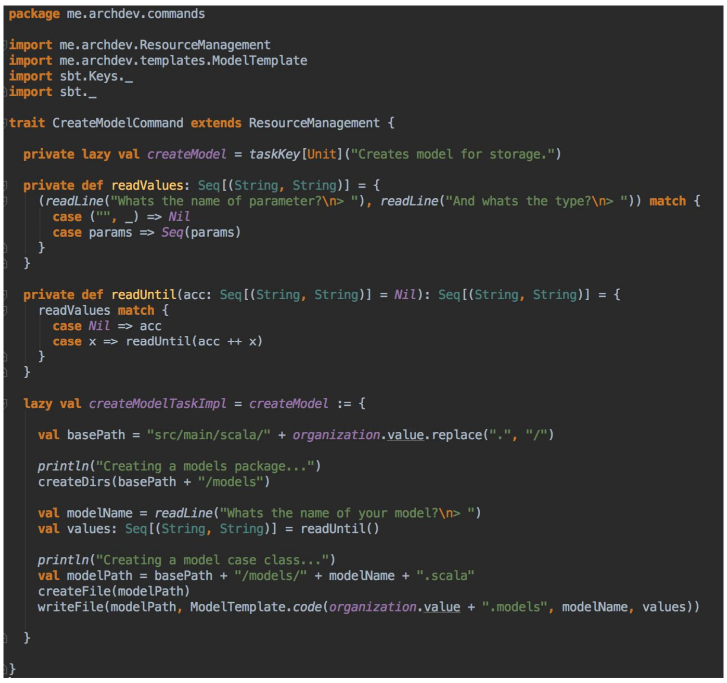
Figure 6 – The Example of service with basic actions of REST service

actions and we execute them by means of the db.run function (fig. 6). Having created all necessary actions for operation with a data model we connect them to external references to service. For this purpose we develop a class of routing which will describe the REST protocol for model.

As conditions, in syntax of akka-http library directives appear. We use the following directives: get, post, pathPrefix, pathEndOrSingleSlash, delete, put. PathPrefix are used for check of authenticity of some part of a way in an external reference and for reading of a part of a way to a routing code. PathEndOrSingleSlash checks whether there came the end of the link. Get, post, put, delete check whether the request belongs to one of these types. An important part of operation of routing is the process description of serialization and deserialising of data. As for exchange with the client we use the JSON format, we need to find a method of creation of internal objects on the basis of the user's request. The spray-json library is for this purpose used. On the basis of a macro of jsonFormatN (Type) it creates functions for serialization and deserialising of objects. Further we add support of this library to ours routing and we have an opportunity to use commands _, toJson and as [Type] for conversion of data. After that the user of a framework needs to add the created routing to a global mapping.

Creation of templates and commands for generation of components

Having all necessary components of system, we will develop templates for generation. They will be executed in the form of objects with the code (params) function. As parameters necessary data for generation of this or that component will be transferred. Having selected dependent parts from the created components we can turn them into variables which will be filled depending on external parameters. Having created all necessary templates we pass to writing of commands for sbt (fig. 7). For reading from command line fixed function of readLine is used. For record in the console



```scala
package me.archdev.commands

import me.archdev.ResourceManagement
import me.archdev.templates.ModelTemplate
import sbt.Keys._
import sbt._

trait CreateModelCommand extends ResourceManagement {

  private lazy val createModel = taskKey[Unit]("Creates model for storage.")

  private def readValues: Seq[(String, String)] = {
    (readLine("Whats the name of parameter?\n> "), readLine("And whats the type?\n> ")) match {
      case ("", _) => Nil
      case params => Seq(params)
    }
  }

  private def readUntil(acc: Seq[(String, String)] = Nil): Seq[(String, String)] = {
    readValues match {
      case Nil => acc
      case x => readUntil(acc ++ x)
    }
  }

  lazy val createModelTaskImpl = createModel := {

    val basePath = "src/main/scala/" + organization.value.replace(".", "/")

    println("Creating a models package...")
    createDirs(basePath + "/models")

    val modelName = readLine("Whats the name of your model?\n> ")
    val values: Seq[(String, String)] = readUntil()

    println("Creating a model case class...")
    val modelPath = basePath + "/models/" + modelName + ".scala"
    createFile(modelPath)
    writeFile(modelPath, ModelTemplate.code(organization.value + ".models", modelName, values))

  }

}
```

Figure 7 – The Example of a command for generation of model

the println function is used. Using a resource manager we write the generated template in the file of a class. To register a command function from sbt packet - taskKey is used.

Conclusion. As a result of the conducted researches the framework for creation of RESTful of web services was offered. The developed framework allows to create with ease new web services (thanks to a set of the built-in components). Each component is an independent part and is easily combined with other parts. Also, the generator of a code which allows to create controllers for access to data models is developed for optimization of operation. The functionality of the developed framework allows to create the complex web applications partitioned into a set of services. At the same time each of them is created on the basis of ready components.

SOURCES

1. Leonard Richardson. RESTful Web Services. [Text] / Richardson L. - O'Reilly Media. - 2007 - 454 p.
2. Bessis N. Development of Distributed Systems from Design to Application and Maintenance [Text]/N.Bessis.- 2012 - 367p.
3. Sam Newman. Building Microservices [Text] / Newman S. - O'Reilly Media. -2015 - 280 p.
4. Odersky M. Programming in Scala: A Comprehensive Step-by-Step Guide [Text]/M.Odersky.- 2011 – 883p.
5. The interactive build tool [Електронний ресурс]/Режим доступу: http://www.scala-sbt.org.