УДК 004.75

В.В. Спірінцев, А.А. Кушка

ФРЕЙМВОРК НА БАЗІ МОВИ ПРОГРАМУВАННЯ SCALA ДЛЯ СТВОРЕННЯ RESTFUL WEB-СЕРВІСІВ

Анотація. Пропонується фреймворк на базі мови програмування Scala для розробки стандартизованих RESTful веб-сервісів. Проект націлений на застосовування іншими розробниками програмного забезпечення для полегшення розробки веб-додатків, створених за допомогою архітектури мікросервісів.

Ключові слова: фреймворк, веб-сервіси, Scala, REST.

Постановка проблеми. Одним з найбільш важливих питань розвитку веб-додатків є процес взаємодії клієнта із сервером. Довгий час цей процес був довільним для кожного додатку. Така ситуація проіснувала до створення перших протоколів серіалізації даних та механізмів для створення довільних запитів. Поява стандартів для комунікації між клієнтом та сервером дозволила вирішити проблему створення різних клієнтів для одного і того ж серверу. Слід зазначити, що на даний момент існує ряд проблем при створенні стандартизованої та ефективної серверної частини: високий поріг входу для реалізації подібних систем (пов'язаний з відсутністю комплексних інструментів для проектування та реалізації веб-сервісів); наявність певного досвіду в розробці веб-додатків (RESTful веб-сервіси доводиться збирати з окремих компонентів [1]); відсутність або мала кількість прикладів організації веб-сервісів в додатку (пов’язано з закритістю вихідних кодів або навіть будь-яких подробиць реалізації). Тому дослідження в напрямку розробки серверної частини на основі веб-сервісів є актуальними.

Аналіз останніх досліджень. Зараз спостерігається динамічне зростання аудиторії мережі Internet. Для надання якісних послуг користувачам інфраструктура сучасних веб-додатків повинна легко та вчасно масштабуватися. Але при збільшенні проекту стає все складніше дотримуватися стабільності додатку, а швидкість написання нового функціоналу знижується. Вирішенням цієї проблеми є роз-

 ISSN 1562-9945

биття додатку на веб-сервіси, але це збільшує кількість несподіваних ситуацій (проблеми з мережею, неочікуване змінення протоколу передачі даних і т.ін.) [2, 3]. Для їх запобігання необхідно використовувати інструменти, які попередять розробника про наявність помилок. Ефективним засобом для цього є вибір інфраструктури JVM та мови програмування Scala [4], яка має потужний статичний аналізатор та підтримку як об'єктно-орієнтовної парадигми так і функціональної парадигми програмування.

Метою статті є створення фреймворку для швидкої розробки RESTful веб-сервісів на базі мови програмування Scala, який організує роботу між компонентами системи та вирішуватиме наступні проблеми: створення структури додатка; уніфікації взаємодії з веб-сервісом; організації асинхронної роботи веб-сервісу; налаштування кешування сервісу; збірки готового додатку і доставки його на сервер.

Основна частина. Для створення фреймворку була обрана мова програмування Scala. З доступних для JVM платформи інструментів складання коду була обрана система sbt [5]. В якості робочого середовища використовується комп'ютер на базі ОС OS X з використанням віртуальної машини Java версії 8 та Scala компілятору версії 2.11.8. В якості редактору коду обрана IntelijIdea 2016.1.

На рис. 1 наведено структуру фреймворку.

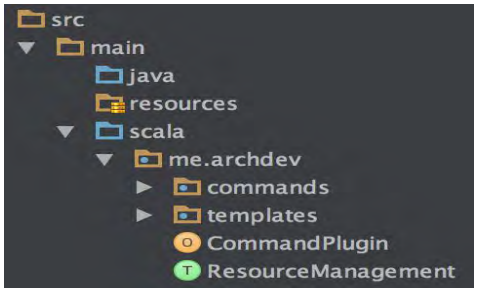

Рисунок 1 - Структура фреймворку

Папка src в корені проекту містить усі сирцеві коди фреймворку, папка main - сирцеві коди основного додатку; папка resources призначена для статичних файлів та scala - для коду додатку. Commands містить класи для реалізації основного функціоналу фреймворку, templates - основи для класів, які фреймворк буде створювати під час роботи. Створений клас CommandPlugin використовується в якості агрегатора усіх доступних команд та є основним класом плагіну. ResourceManagement є класом для взаємодії з файловою системою користувача плагіну.

Налаштовуємо інструмент складання, щоб мати можливість створювати снапшоти для тестування. Для цього у корені проекту створюємо файл build.sbt та налаштовуємо його параметри.

Створення менеджеру ресурсів. Одним з основних класів для фреймворку є менеджер ресурсів (використовується для створення та наповнення файлів з кодом програми користувача). Ця задача вирішується завдяки пакету java.nio. Особливістю роботи з файлами у пакеті java.nio є те, що взаємодія з даними у файловій системі відбувається за допомогою буферів. Завдяки буферам є можливість працювати з великими файлами, послідовно зчитуючи та оброблюючи їх частини. Для створення фреймворку необхідні такі операції як: створення дерев каталогів, створення файлу, читання та запис у файл, запис у кінець файлу та копіювання. Сам клас буде розроблений у форматі трейту – це інтерфейс який може мати базову реалізацію. Основною особливістю його є те, що клас може успадковуватися від безлічі трейтів.

Створення класів для запуску веб-сервера. Для того, щоб користувач фрейворку мав можливість одразу після створення проекту запустити веб-сервер, нам необхідно створити базовий клас. Далі налаштовуємо та запускаємо веб-сервер. В якості веб-серверу буде використана бібліотека akka-http, яка надає нам http-server написаний на мові Scala. Обраний веб-сервер є асинхронним та багато поточним. Його основною конфігурацією є файл з роутингом запитів, у ньому користувач фреймворку має описати структуру посилань для доступу до RESTful ресурсів. Після створення функції для генерації базових класів ми маємо призначити команду фреймворка для виклику цієї функції з консолі інструменту складання sbt. Для цього використовуємо TaskKey з назвою provision. Результатом роботи буде створення двух класів у стандартному пакеті користувача: для опису роутингу, для запуску веб-сервісу.

Налаштування роботи з базою даних. Однією з найважливіших особливостей RESTful веб-сервісів є відсутність внутрішнього стану. Це означає те, що усі дані повинні зберігатися окремо, тобто у базах даних. Використовуємо JDBC драйвер для надання доступу до бази даних користувача фреймворку. Для того, щоб полегшити розробку веб-сервіса на створеному фреймворку додаємо ще один рівень абстракції для генерації SQL коду. Для цього буде використана бібліотека мови програму-

вання Scala – Slick. Особливістю цієї бібліотеки є те, що вона призводить конвертацію Scala коду у SQL під час компіляції проекту. Завдяки цьому результат роботи завжди перевіряється статичним аналізатором мови Scala, а функціонал бібліотеки не впливає на швидкість виконання коду як у випадку з аналогами на мові Java. Зі сторони фреймворку ми повинні надати користувачу спосіб використовувати вже зконфігуровану бібліотеку Slick у своєму коді. Для цього створюємо трейт DatabaseConfig, який буде доданий до кожного класу користувача, де буде виконуватись робота з базою даних.

Створення моделі даних. Основою RESTful веб-сервісу є модель даних з якими він буде працювати. Для цього створюємо команду, що буде збирати дані користувача про необхідні моделі та генерувати клас з необхідними даними для зберігання цієї моделі у базі даних. Для того, щоб задіяти бібліотеку Slick, підключимо вже створений трейт DatabaseConfig до створеного трейту UserTable в якому імпортуємо синтаксис бібліотеки за допомогою команди import driver.api._. Для створення маппінгу між даними та таблицею в базі даних створюємо клас, який буде наслідуватися від класу Table. В середині цього класу описуємо поля за допомогою функції column[Type](fieldname, flags). Після цього створюємо аплікативну функцію яка міститиме у собі інформацію про співвідношення полів класу до полів у базі даних. При створенні аплікативної функції надаємо бібліотеці набір полів у базі даних та функції конвертації класу у набір даних та набору даних у клас. Ці функції вже надані нам мовою програмування Scala. Після цього створюємо набір базових дій з базою даних за допомогою макросу TableQuery[Type]. Цей макрос отримує дані мапінгу моделі до бази даних та генерує базові операції над таблицею.

Створення базових дій REST сервісу та створення REST роутингу для роботи з сервісом. Маючи модель для зберігання даних та базові дії над базою даних ми можемо згенерувати сервіс який буде надавати базові дії для REST сервісу. Для цього згенеруємо трейт який унаслідується від трейту з описом маппінгу між базою даних та моделлю, та реалізуємо операції які будуть: виводити усі записи, виводити окремий запис на основі його ідентифікаційного коду, додавати запис, оновлювати та видаляти записи. Для імплементації цих дій будемо використовувати бібліотеку Slick. За допомогою вже створених макросом базових дій з базою даних, описуємо необхідні дії та виконуємо їх за

допомогою функції db.run. Створивши усі необхідні дії для роботи з моделлю даних зв'язуємо їх з зовнішніми посиланнями на сервіс. Для цього розробляємо клас роутингу, який опише REST протокол для моделі. В якості умов, в синтаксисі бібліотеки akka-http виступають директиви. Використовуємо наступні директиви: get, post, pathPrefix, pathEndOrSingleSlash, delete, put. PathPrefix використовуються для перевірки існування деякої частини шляху у зовнішньому посиланні та для зчитування частини шляху у код роутингу. PathEndOrSingleSlash перевіряє чи вже настав кінець посилання. Get, post, put, delete перевіряють чи запит відноситься до одного з цих типів. Важливою частиною роботи роутингу є опис процесу серіалізації та десеріалізації даних. Оскільки для обміну з клієнтом ми використовуємо формат JSON, то нам необхідно знайти спосіб створення внутрішних об'єктів на основі запиту користувача. Для цього використовується бібліотека spray-json. На основі макросу jsonFormatN(Type) вона генерує функції для серіалізації та десеріалізації об'єктів.

Надалі ми додаємо підтримку цієї бібліотеки до нашого роутингу і отримуємо можливість використовувати команди _,toJson та as[Type] для конвертації даних. Після цього користувачу фреймворку залишається додати створений роутинг до глобального маппінгу.

Створення шаблонів та команд для генерації компонентів. Маючи усі необхідні компоненти системи, розробимо шаблони для генерації. Вони будуть виконані у вигляді об'єктів з функцією code(params). В якості параметрів будуть передаватися необхідні дані для генерації того чи іншого компоненту. Виділив залежні частини зі створених компонентів ми можемо перетворити їх на змінні, що будуть заповнені в залежності від зовнішніх параметрів. Створивши усі необхідні шаблони переходимо до написання команд для sbt (рис.2). Для читання з командної строки використовується стандартна функція readLine. Для запису у консоль використовується функція println. Використовуючи менеджер ресурсів записуємо згенерований шаблон у файл класу. Для того, щоб зареєструвати команду використовується функція з пакету sbt - taskKey.

 ISSN 1562-9945

```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))

  }

}
```

Рисунок 2 – Приклад команди для генерації моделі

Висновки. В результаті проведених досліджень було запропоновано фреймворк для створення RESTful веб-сервісів, який дозволяє створювати нові веб-сервіси завдяки набору вбудованих компонентів. Кожен компонент є незалежною частиною та легко комбінується з іншими частинами. Також, для оптимізації роботи розроблено генератор коду, що дозволяє створювати контролери для доступу до моделей даних. Функціонал розробленого фреймворку дозволяє створювати комплексні веб-додатки, поділені на безліч сервісів. При цьому кожен з них створюється на основі готових компонентів.

ЛІТЕРАТУРА

1. Richardson L. RESTful Web Services. [Text] / L. Richardson - 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. Newman S. Building Microservices [Text] / S. Newman - 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 [Ел.ресурс]/Реж.доступу: http://www.scala-sbt.org.