JavaCro'15

Biblioteka Slick

Ivan Senji

identalia
CONSULTING

Slick

JavaCro'15

1. Uvod
2. Zašto Slick?
3. Zašto Slick 3?
4. Kako započeti
5. Kako i što radi Slick
6. Funkcionalnosti kroz primjere

Uvod

JavaCro'15

Slick: Functional Relational Mapping for Scala

http://slick.typesafe.com/

Typesafe Reactive Platform

A Unified Platform for Building Modern Apps

- *play*
- *akka*
- *Scala*
- *Slick*

play

JavaCro'15

The High Velocity Web Framework For Java and Scala

akka

Akka is a toolkit and runtime for building highly concurrent, distributed, and resilient message-driven applications on the JVM.

Uvod

JavaCro'15

Scala

Rad s bazom iz Scala koda

Type Safe

Statička provjera svih tipova prilikom kompajliranja

Composable

Slaganje upita iz dijelova koristeći operacije nad Scala kolekcijama

Zašto Slick?

Zašto Slick?

JavaCro'15

Nije objektno-relacijsko mapiranje

Nema promjenjivih grafova objekata

Fokus na pristupu relacijskim bazama

Zašto još Slick?

JavaCro'15

Koncepti relacijskih baza preslikani na koncepte u Scali

Jednostavna konfiguracija:

- bez xml-a
- malo Scala koda

Precizna kontrola nad podacima

Zašto Slick 3?

JavaCro'15

Staro: api za rad s bazom *deprecated*

- bit će izbačen u verziji 3.1

Novo: *monadic database I/O actions API*

- monad je pojam koji predstavlja izračun (akciju) i definira pravila za povezivanje izračuna (akcija)
- akcije se izvršavaju asinkrono

Što je Monad

JavaCro'15

- puno odgovora / tutorijala / objašnjenja
- http://stackoverflow.com/questions/44965/what-is-a-monad
- http://james-iry.blogspot.com/2007/09/monads-are-elephants-part-1.html
- http://blog.sigfpe.com/2006/08/you-could-have-invented-monads-and.html
- ...
- monad je standardno sučelje prema različitim podatkovnim i kontrolnim strukturama

JavaCro'15

Kako započeti sa Slickom

Kako započeti sa Slickom

JavaCro'15

Typesafe Activator

Dodavanje Slick-a u postojeći projekt

sbt:

```
libraryDependencies ++= Seq (
  "com.typesafe.slick" %% "slick" % "3.0.0-RC3"
)
```

maven:

```
<dependency>
  <groupId>com.typesafe.slick</groupId>
  <artifactId>slick_2.10</artifactId>
  <version>3.0.0-RC3</version>
</dependency>
```

Kako započeti - konfiguracija

JavaCro'15

Logiranje: slf4j + implementacija

- logback

```
<logger name="slick" level="INFO"/>
<logger name="slick.jdbc.StatementInvoker.result" level="DEBUG"/>
<logger name="slick.jdbc.JdbcBackend.statement" level="DEBUG"/>
```

Konfiguracija baze

- Typesafe Config
- JDBC URL
- DataSource
- JNDI

Typesafe Config

JavaCro'15

https://github.com/typesafehub/config

"Configuration library for JVM languages."

application.conf:

```
dbconf = {
  url = "jdbc:postgresql://localhost:5432/db1?user=me&password=m"
  driver = org.postgresql.Driver
}
```

scala:

```
val db = Database.forConfig("dbconf")
```

JavaCro'15

JDBC URL

```
val db =
  Database.forURL(
    "jdbc:postgresql://localhost:5432/db1?user=me&password=m",
    driver="org.postgresql.Driver")
```

DataSource

```
val db = Database.forDataSource(dataSource: javax.sql.DataSource)
```

JNDI

```
val db = Database.forName(jndiName: String)
```

Generiranje koda

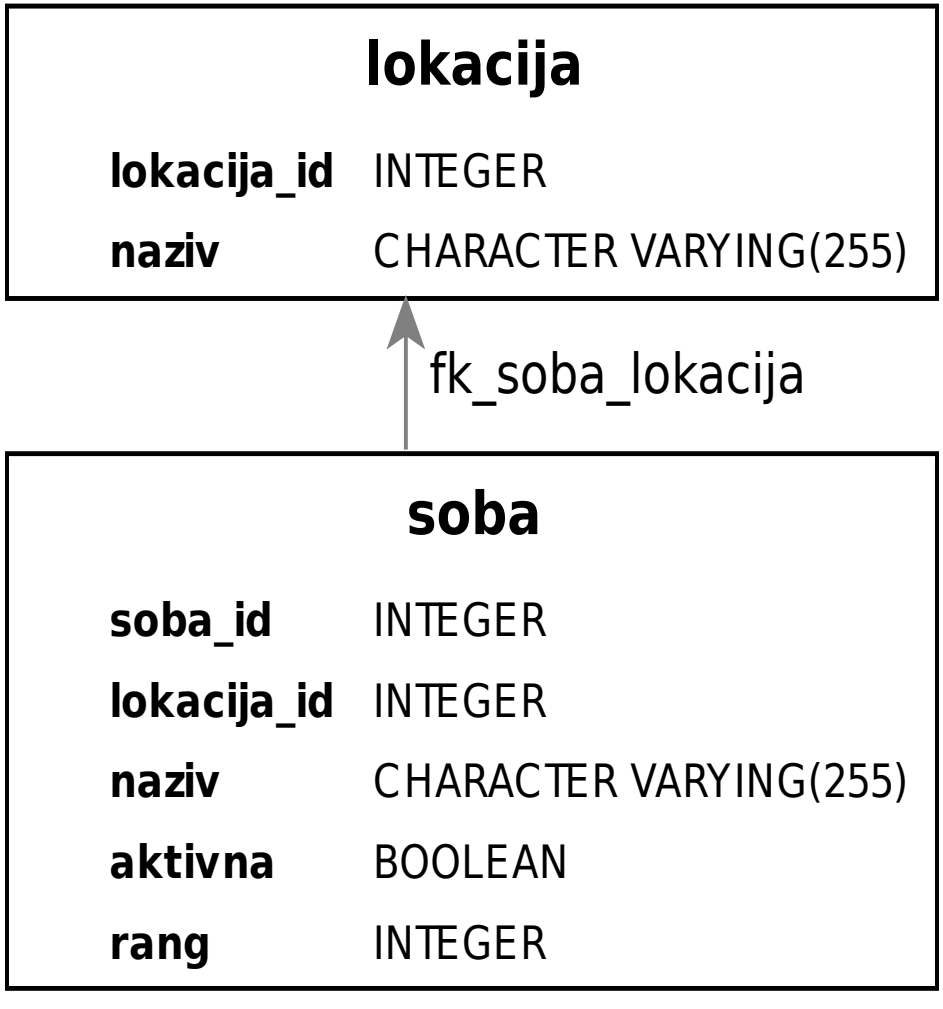
JavaCro'15

```
// akcija za dohvat modela
val modelAction = PostgresDriver.createModel(Some(PostgresDriver.defaultTables))
val modelFuture = db.run(modelAction)

val codegenFuture = modelFuture.map(model => new SourceCodeGenerator(model) {

  override def tableName = { dbTableName =>
    dbTableName.toLowerCase.toCamelCase
  }

  override def Table = new Table(_) {
    table =>
    // ...
  }
})

codegenFuture.onSuccess { case codegen =>
  codegen.writeToFile(
    "slick.driver.PostgresDriver", "/home/ivan/Projects/JavaCro2015Project/src/main/scala/",
    "hr.ivan", "Tables", "Tables.scala"
  )
}

scala.concurrent.Await.ready(codegenFuture, Duration.Inf)
```

Primjeri

Primjer - DB model

JavaCro'15



Primjer - LOKACIJA (1)

JavaCro'15

```
CREATE TABLE lokacija (
    lokacija_id   SERIAL
  , naziv         VARCHAR(255) NOT NULL
  , CONSTRAINT pk_lokacija PRIMARY KEY (lokacija_id)
);
```

```
type LokacijaRow = (Int, String)
```

```
class Lokacija(_tableTag: Tag)
      extends Table[LokacijaRow](_tableTag, "lokacija") {

  val lokacijaId: Rep[Int] =
      column[Int]("lokacija_id", O.AutoInc, O.PrimaryKey)
  val naziv: Rep[String] =
      column[String]("naziv", O.Length(255,varying=true))

  def * = (lokacijaId, naziv)
}
```

```
lazy val Lokacija = new TableQuery(tag => new Lokacija(tag))
```

Primjer - LOKACIJA (2)

JavaCro'15

```
CREATE TABLE lokacija (
    lokacija_id   SERIAL
  , naziv         VARCHAR(255) NOT NULL
  , CONSTRAINT pk_lokacija PRIMARY KEY (lokacija_id)
);
```

```
case class LokacijaRow(lokacijaId: Int, naziv: String)
```

```
class Lokacija(_tableTag: Tag)
      extends Table[LokacijaRow](_tableTag, "lokacija") {

  val lokacijaId: Rep[Int] =
      column[Int]("lokacija_id", O.AutoInc, O.PrimaryKey)
  val naziv: Rep[String] =
      column[String]("naziv", O.Length(255,varying=true))

  def * = (lokacijaId, naziv) <>
          (LokacijaRow.tupled, LokacijaRow.unapply)
}
```

```
lazy val Lokacija = new TableQuery(tag => new Lokacija(tag))
```

tupled i unapply


```
scala> case class LokacijaRow(lokacijaId: Int, naziv: String)
defined class LokacijaRow

scala> LokacijaRow.tupled
res0: ((Int, String)) => LokacijaRow = <function1>

scala> LokacijaRow.unapply _
res2: LokacijaRow => Option[(Int, String)] = <function1>
</function1></function1>
```

Rep

```
trait slick.lifted.Rep[T]
// Common base trait for all lifted values, including columns.
```

http://slick.typesafe.com/doc/3.0.0-RC3/api/index.html#slick.lifted.Rep

Tag

JavaCro'15

```
// A Tag marks a specific row represented
// by an AbstractTable instance.
```

http://slick.typesafe.com/doc/3.0.0-RC3/api/index.html#slick.lifted.Tag

TableQuery

```
class TableQuery[E <: AbstractTable[_]]
      extends Query[E, lifted.TableQuery.E.TableElementType, Seq]
// Represents a database table.
```

http://slick.typesafe.com/doc/3.0.0-RC3/api/index.html#slick.lifted.TableQuery

Primjer - SOBA

JavaCro'15

```
CREATE TABLE soba (
    soba_id      SERIAL
  , lokacija_id INTEGER NOT NULL
  , naziv        VARCHAR(255) NOT NULL
  , aktivna      BOOLEAN DEFAULT true NOT NULL
  , rang         INTEGER NOT NULL  -- redoslijed sortiranja
  , CONSTRAINT pk_soba PRIMARY KEY (soba_id)
  , CONSTRAINT fk_soba_lokacija
               FOREIGN KEY (lokacija_id)
               REFERENCES lokacija (lokacija_id)
);
```

```
case class SobaRow (
  sobaId: Int,
  lokacijaId: Int,
  naziv: String,
  aktivna: Boolean = true,
  rang: Int)
```

Primjer - SOBA

JavaCro'15

```
CREATE TABLE soba (
    soba_id     SERIAL
  , lokacija_id INTEGER NOT NULL
  , naziv       VARCHAR(255) NOT NULL
  , aktivna     BOOLEAN DEFAULT true NOT NULL
  , rang        INTEGER NOT NULL  -- redoslijed sortiranja
  , CONSTRAINT pk_soba PRIMARY KEY (soba_id)
  , CONSTRAINT fk_soba_lokacija
               FOREIGN KEY (lokacija_id)
               REFERENCES lokacija (lokacija_id)
);
```

```
class Soba(_tableTag: Tag) extends Table[SobaRow](_tableTag, "soba") {
  def * = (sobaId, lokacijaId, naziv, aktivna, rang) <>
          (SobaRow.tupled, SobaRow.unapply)

  val sobaId: Rep[Int] = column[Int]("soba_id", O.AutoInc, O.PrimaryKey)
  val lokacijaId: Rep[Int] = column[Int]("lokacija_id")
  val naziv: Rep[String] = column[String]("naziv", O.Length(255,varying=true))
  val aktivna: Rep[Boolean] = column[Boolean]("aktivna", O.Default(true))
  val rang: Rep[Int] = column[Int]("rang")

  lazy val lokacijaFk = foreignKey("fk_soba_lokacija", lokacijaId, Lokacija)
    (r => r.lokacijaId, onUpdate=ForeignKeyAction.NoAction,
    onDelete=ForeignKeyAction.NoAction)
}
```

Scala

JavaCro'15

insert

```
Lokacija += LokacijaRow(0, "Rovinj")
```

batch insert

```
Lokacija ++= Seq (
  LokacijaRow(0, "Lokacija 1"), LokacijaRow(0, "Lokacija 2"),
  LokacijaRow(0, "Lokacija 3"), LokacijaRow(0, "Lokacija 4")
)
```

query

```
Lokacija.map(_.naziv)
```

update

```
Lokacija.map(_.naziv.toUpperCase)
```

Type Safe

JavaCro'15

```
val nazivi : DBIO[Seq[String]] = Lokacija.map(_.naziv).result
```

```
def spremiLokaciju(naziv: String): DBIO[Int] =
  (Lokacija returning Lokacija.map(_.lokacijaId)) +=
    LokacijaRow(0, naziv)
```

```
val poredaneLokacije : DBIO[Seq[LokacijaRow]] =
  Lokacija.sortBy(_.naziv).result
```

Composable

JavaCro'15

```
Lokacija.filter(_.lokacijaId <= 10)
        .map ( l =>
          l.lokacijaId.asColumnOf[String] ++ "-" ++ l.naziv
        )
        .sortBy(naziv => naziv)
        .take(10)
```

```
val lokacijaById = Lokacija.filter(_.lokacijaId === 1)

lokacijaById.map (_.naziv)
            .update("Rovinj, JavaCro 2015")
```

Od upita do rezultata

JavaCro'15

Query ➔ Action ➔ Future

```
val q = for (l <- Lokacija) yield l.naziv
val a: DBIO[Seq[String]] = q.result
val f: Future[Seq[String]] = db.run(a)

f.onSuccess { case s => println(s"Rezultat: $s") }
```

q: kreiranje upita

a: kreiranje/kombiniranje akcija

f: izvršavanje akcije i dobivanje rezultata u budućnosti

DBIO

JavaCro'15

```
type DBIO[+R] = DBIOAction[R, NoStream, Effect.All]
```

*A **Database I/O Action** that can be executed on a database.*
*The DBIOAction type allows a separation of **execution logic** and **resource usage management logic** from **composition logic**.*
*DBIOActions can be composed with methods such as **andThen**, **andFinally** and **flatMap**.*

http://slick.typesafe.com/doc/3.0.0-RC3/api/index.html#slick.dbio.DBIOAction

DBIOAction: nizanje operacija

JavaCro'15

```
val a1 = for {
  _ <- schema.create
  _ <- Lokacija += LokacijaRow(0, "Rovinj")
  _ <- Lokacija += LokacijaRow(0, "Zagreb")
  lokacije <- Lokacija.result
} yield (lokacije)
```

ili

```
val a2 =
  schema.create >>
  (Lokacija += LokacijaRow(0, "Rovinj")) >>
  (Lokacija += LokacijaRow(0, "Zagreb")) >>
  Lokacija.result
```

ili

```
val a3 = DBIO.seq (
  schema.create,
  Lokacija += LokacijaRow(0, "Rovinj"),
  Lokacija += LokacijaRow(0, "Zagreb"),
  Lokacija.result
)
```

>>

JavaCro'15

```
final def >> [R2] (a: DBIOAction[R2]): DBIOAction[R2] =
  andThen[R2](a)
```

andThen

```
/* Izvrši drugu akciju nakon ove akcije i vrati rezultat druge
   akcije. Greška je ako bilo koja od te dvije akcije završi s
   greškom */
def andThen[R2](a: DBIOAction[R2]): DBIOAction[R2] =
    AndThenAction[R2](this, a)
```

flatMap

```
/* Uspješan rezultat trenutne akcije koristi za izvršavanje
   druge akcije. */
def flatMap[R2](f: R => DBIOAction[R2]): DBIOAction[R2] =
    FlatMapAction[R2](this, f, executor)
```

map

JavaCro'15

```
/* Transformiraj uspješan rezultat izvršavanja akcije. */
def map[R2](f: R => R2): DBIOAction[R2] =
    flatMap[R2](r => SuccessAction[R2](f(r)))
```

zip

```
/** Izvrši drugu akciju nakon trenutne i vrati rezultat
    obje akcije. */
def zip[R2](a: DBIOAction[R2]): DBIOAction[(R, R2)] =
  SequenceAction[Any](Vector(this, a)).map { r =>
    (r(0).asInstanceOf[R], r(1).asInstanceOf[R2])
}
```

Izvršavanje SQL upita

JavaCro'15

sqlu

DML statementi koji vraćaju broj redaka kao rezultat

```
def insert(l: LokacijaRow): DBIO[Int] =
  sqlu"insert into lokacija values (${l.naziv})"
```

Izvršavanje SQL upita

JavaCro'15

sql

DML statement koji vraća ResultSet

```
sql"""select l.lokacija_id, l.naziv
      from lokacija""".as[(Int, String)]
```

ili

```
implicit val getLokacijaResult =
    GetResult(r => LokacijaRow(r.<<, r.<<))

sql"""select l.lokacija_id, l.naziv
      from lokacija""".as[LokacijaRow]
```

SQL - korisnički definirana funkcija

JavaCro'15

funkcija u bazi:

```
CREATE OR REPLACE FUNCTION "public"."sha256" (bytea)  RETURNS text
  IMMUTABLE
  RETURNS NULL ON NULL INPUT
AS $$
SELECT encode(digest($1, 'sha256'), 'hex')
$$ LANGUAGE sql
```

deklaracija funkcije (Slick):

```
val sha256 = SimpleFunction.unary[Array[Byte],String]("sha256")
```

korištenje funkcije:

```
val a = Lokacija.map { l =>
          sha256(l.naziv.asColumnOf[Array[Byte]])
        }.result
```

Join

Join - aplikativni

JavaCro'15

```
val crossJoin: Query[_, (LokacijaRow, SobaRow), Seq] = for {
  (l, s) <- Lokacija join Soba
} yield (l, s)
```

```
val innerJoin: Query[_, (LokacijaRow, SobaRow), Seq] = for {
  (l, s) <- Lokacija join Soba on
              (_.lokacijaId === _.lokacijaId)
} yield (l, s)
```

```
val leftOuterJoin: Query[_, (LokacijaRow, Option[SobaRow]), Seq] =
  for {
    (l, s) <- Lokacija joinLeft Soba on
                (_.lokacijaId === _.lokacijaId)
  } yield (l, s)
```

```
val rghtOuterJoin: Query[_, (Option[LokacijaRow], SobaRow), Seq] =
  for {
    (l, s) <- Lokacija joinRight Soba on
                (_.lokacijaId === _.lokacijaId)
  } yield (l, s)
```

Join - monadski

JavaCro'15

```
val crossJoin = for {
  l <- Lokacija
  s <- Soba
} yield (l, s)
```

```
val innerJoin = for {
  l <- Lokacija
  s <- Soba if l.lokacijaId === s.lokacijaId)
} yield (l, s)
```

Pitanja?