# DSL in JVM

## The Good, The Bad and The Ugly

Андрей Михеев

Jjoy @ Рязань
14 июня 2017

Предметно-ориентированный язык — это язык программирования с ограниченными выразительными возможностями, ориентированный на некую конкретную предметную область

Мартин Фаулер

# Виды DSL

- Внешний (SQL, CSS, HTML)
- Внутренний (Hibernate, Slick, Linq)

# Зачем?

- DSL делает ваш код читабельнее и короче
- DSL избавляет от формализма командную работу
- DSL увеличивает скорость разработки
- DSL увеличивает коэффициент повторного использования кода
- DSL позволяет получить программные системы, расширяемые их пользователями

# Недостатки

- стоимость проектирования, реализации и сопровождения может быть достаточно велика
- необходимость дополнительного обучения пользователей
- в некоторых случаях область действия DSL определить довольно трудно
- сложность сохранения равновесия между конструкциями, использующимися в DSL, и конструкциями языка программирования общего назначения

# Особенности DSL

- Ограничен базовым языком
- Неполнота – на DSL нельзя писать программы
- Зависимость от внешней системы – браузера, интерпретатора
- DSL бывают как декларативные, так и императивные

# На самом деле

Плох тот проект, который не мечтает о своем DSL

# Немного о Java

- XML
- ООП
- Аннотации
- Fluent API

# XML

```
<bean id="testBean" class="org.springframework.beans.TestBean" scope="prototype">
  <property name="age" value="10"/>
  <property name="spouse">
    <bean class="org.springframework.beans.TestBean">
      <property name="age" value="11"/>
    </bean>
  </property>
</bean>
```

# XML

```xml
<?xml version="1.0" encoding="UTF-8"?>
<FSM>
    <STATE id="NEW" type="ID">
        <MESSAGE id="SIGN" action="sign" nextState="SIGNED" />
        <MESSAGE id="DELETE" action="delete" nextState="DELETED" />
        <MESSAGE id="ARCHIVE" action="archive" nextState="ARCHIVED" />
    </STATE>
    <STATE id="SIGNED">
        <MESSAGE id="ARCHIVE" action="archive" nextState="ARCHIVED" />
    </STATE>
    <STATE id="DELETED" />
    <STATE id="ARCHIVED" />
</FSM>
```

# ООП

```
State NEW = new State("New");
State SIGNED = new State("Signed");
State DELETED = new State("Deleted");
State ARCHIVED = new State("Archived");
satetes.add(NEW);
satetes.add(SIGNED);
satetes.add(DELETED);
satetes.add(ARCHIVED);

Event SIGN = new Event("Sign");
Event DELETE = new Event("Delete");
Event ARCHIVE = new Event("Archive");
events.add(SIGN);
events.add(DELETE);
events.add(ARCHIVE);

transitions.add(new Transition(Events.SIGN).from(NEW).to(SIGNED));
transitions.add(new Transition(Events.DELETE).from(NEW).to(DELETED));
transitions.add(new Transition(Events.SIGN).from(OPEN).to(ARCHIVED));

StateMachine stateMachine = new StateMachine(satetes, events, transitions);
stateMachine.start(NEW);
stateMachine.process(SIGN);
```

# Аннотации

```
@StateMachine
@Configurable("exampleStateMachine")
public class ExampleStateMachineImpl implements ExampleStateMachine {

    static private Log log = LogFactory.getLog(StateMachine.class.getName());
    private int numExecutions = 0;
    static int NumberExecutions = 3;

    @StateTransition(from = { "" }, to = "new")
    public void execute() {
        log.info("state machine started");

        sign();
    }

    @StateTransition(from = { "new", "signed" }, to = "archived")
    public void sign() {
        log.info("state machine signed");
        numExecutions++;
        if (numExecutions >= NumberExecutions) {
            deleted();
        } else {
            sign();
        }
    }

    @StateTransition(from = { "archived" }, to = "deleted")
    public void deleted() {
        log.info("state machine finished");
    }
}
```

# Fluent API

```
StateMachine stateMachine = new StateMachine.Builder<State, Event, RuntimeState>()
    .initialState(NEW)
    .onState(NEW, s -> s
            .onEvent(SIGN).changeTo(SIGNED).noAction()
            .onEvent(DELETE).changeTo(DELETED).noAction()
            .onEvent(ARCHIVE).changeTo(ARCHIVED).noAction()
            .onExitAction(RuntimeState::exitInitialState))
    .onState(SIGNED, s -> s
            .onEntryAction(Utils::signDoc)
            .onEvent(ARCHIVE).changeTo(ARCHIVED).noAction()
    .onState(DELETED, s -> s
            .onEntryAction(RuntimeState::deleteDoc)
    .onState(ARCHIVED, s -> s
            .onEntryAction(RuntimeState::archiveDoc)
    .build();
```

# Groovy, Kotlin, Scala

```
machine name: "exampleStateMachine",
        initialState: NEW,
        documentTypeDesc: DocDesc.name,
        entityClassName: Doc.name,
        states: {
            state nameSys: NEW, name: "Новый"
            state nameSys: SIGNED, name: "Подписан"
            state nameSys: DELETED, name: "Удален"
            state nameSys: ARCHIVED, name: "Архивный"
        },
        actions: {
            action nameSys: ACTION_SIGN,
                    label: "Подписать",
                    from: [NEW],
                    to: SIGNED

        action nameSys: ACTION_DELETE,
                    label: "Удалить",
                    from: [NEW, SIGNED],
                    to: DELETED

        action nameSys: ACTION_ARCHIVE,
                    label: "Архивировать",
                    from: [NEW, SIGNED],
                    to: ARCHIVED
    }
```

# Билдеры

```
html {
    head {
        title {"XML encoding with Groovy"}
    }
    body {
        h1 {"XML encoding with Groovy"}

        // an element with attributes and text content
        a(href = "http://groovylang.org") {"Groovy"}

        // mixed content
        p {
            "This is some"
            b {"mixed"}
            a(href = "http://groovylang.org") {"Groovy"}
            "project"
        }

        // content generated by
        p {
            persons.each { p ->
                li p.name
            }
        }
    }
}
```

```
html {
    head {
        title {+"XML encoding with Kotlin"}
    }
    body {
        h1 {+"XML encoding with Kotlin"}

        // an element with attributes and text content
        a(href = "http://kotlinlang.org") {+"Kotlin"}

        // mixed content
        p {
            +"This is some"
            b {+"mixed"}
            a(href = "http://kotlinlang.org") {+"Kotlin"}
            +"project"
        }
        p {+"some text"}

        // content generated by
        p {
            for (arg in args)
                +arg
        }
    }
}
```

```
Html {
    Head {
        Title {"XML encoding with Scala"}
    }
    Body {
        H1 {"XML encoding with Scala"}

        // an element with attributes and text content
        A(href -> "http://scala-lang.org") {"Scala"}

        // mixed content
        P {
            "This is some"
            B {"mixed"}
            A(href -> "http://scala-lang.org") {"Scala"}
            "project"
        }

        // content generated by
        Ul {
            for(p <- persons) {
              Li{ p.name }
            }
        }
    }
}
```

```
handleMessage(new Handler() {
    @Override
    void handle(String message) {
        println message
    }
})



handleMessage {
    println message
}
```

```
class HTML {
    fun body() { ... }
}

fun html(init: HTML.() -> Unit): HTML {
    val html = HTML() // create the receiver object
    html.init()       // pass the receiver object to the lambda
    return html
}


html {        // lambda with receiver begins here
    body()    // calling a method on the receiver object
}
```

```
html {
    head {
        head {}
    }
    // ...
}
```

```
@DslMarker
annotation class HtmlTagMarker

@HtmlTagMarker
abstract class Tag(val name: String) : Element {...}

abstract class TagWithText(name: String) : Tag(name) {
    operator fun String.unaryPlus() {
        children.add(TextElement(this))
    }
}

class HTML : TagWithText("html") {
    fun head(init: Head.() -> Unit) = initTag(Head(), init)
    fun body(init: Body.() -> Unit) = initTag(Body(), init)
}
class Head : TagWithText("head") {
    fun title(init: Title.() -> Unit) = initTag(Title(), init)
}

class A : BodyTag("a") {
    var href: String
        get() = attributes["href"]!!
        set(value) {
            attributes["href"] = value
        }
}
```

# Расширение существующих классов

# Groovy

```
String.metaClass.withBrackets = {
    "($delegate)"
}

"test".withBrackets()
```

# Kotlin

```
// extension function
fun String.withBrackets(): String {
    return "($this)"
}

"test".withBrackets()
```

# Kotlin

```
// extension properties
val Int.seconds: Duration
        get() = Duration.standartSeconds(this.toLong())

val timeout = 30.seconds
```

# Scala

```
// Неявные преобразования.

implicit def str2extStr(s: String): ExtString = new ExtString(s)

class ExtString(s: String) {
    def withBrackets() = s"($s)"
}

"hello".withBrackets()
```

# Groovy

```
// Перегрузка операторов

class Complex {
    Double real
    Double img

    def plus(Complex c) {
        new Complex(this.real + c.real, this.img + c.img)
    }
}

def c1 = new Complex(1.0, 1.7)
def c1 = new Complex(2.0, 2.4)

println(c1 + c2)


a + b = a.plus(b)
a - b = a.minus(b)
a >> b = a.rightShift(b)
a ++  = a.next()
```

# Kotlin

```
// Перегрузка операторов

class Complex(val real: Double, val img: Double) {
    operator fun plus(c: Complex) = Complex(this.real + c.real, this.img + c
}


def c1 = new Complex(1.0, 1.7)
def c1 = new Complex(2.0, 2.4)

println(c1 + c2)


a + b = a.plus(b)
a - b = a.minus(b)
a * b = a.times(b)
a / b = a.div(b)
a...b = a.range(b)
```

# Scala

```
// Перегрузка операторов

class Complex(real: Double, img: Double) {
   def +(c: Complex) = new Complex(this.real + c.real, this.img + c.img)
   def ->() = ...
   def ->>() = ...
}

def c1 = new Complex(1.0, 1.7)
def c1 = new Complex(2.0, 2.4)

println(c1 + c2)
println(c1 -> c2)
println(c1 ->> c2)
```

# Scala

```
// Перегрузка операторов

class Complex(real: Double, img: Double) {

   def -%##>~>~>() = ...
}

def c1 = new Complex(1.0, 1.7)
def c1 = new Complex(2.0, 2.4)
```

```
println(c1 -%##>~>~> c2)
```

# Kotlin

```
// Объекты

object Tuple {
    operator fun<T> invoke(t: T): Tuple1<T> = Tuple1(t)
    operator fun<T1, T2> invoke(t1: T1, t2: T2): Tuple2<T1, T2> = Tuple2(t1, t2)
    operator fun<T1, T2, T3> invoke(t1: T1, t2: T2, t3: T3): Tuple3...
    ...
}

Tuple(1)
Tuple(1, 2)
Tuple(1, 2, 3)
```

# Scala

```
// Объекты

object Tuple {
    def apply(t: Int) = Tuple1(t)
    def apply(t1: Int, t2: Int) = Tuple2(t1, t2)
    def apply(t1: Int, t2: Int, t3: String) = Tuple3(t1, t2, t2)
}

Tuple(1)
Tuple(1, 2)
Tuple(1, 2, 3)
```

# Groovy

```
// Инфиксная нотация функции

1 plus 2

1.plus(2)
```

# Kotlin

```
// Инфиксная нотация функции

infix fun Int.plus(x: Int): Int {
...
}


1 plus 2

1.plus(2)
```

# Scala

```
// Инфиксная нотация функции

1 + 2
1.+(2)


1 to 5
1.to(5)
```

# Scala

```
val g = Graph(1~2, 2~3, 2~4, 3~5, 4~5)
val h = Graph(3~4, 3~5, 4~6, 5~6)

g union h
g diff h
g intersect h
```

# Scala

```
1.kilograms to Pounds          // Double: 2.2046226218487757

kilogram / pound               // Double: 2.2046226218487757

2.1.pounds to Kilograms        // Double: 0.952543977

2.1.pounds / kilogram          // Double: 0.952543977

100.C to Fahrenheit            // Double: 212.0
```

# Scala

```
Graph((1 ~+#> 2)("A"), (1 ~+#+#> 1)(B))

Route((a |#| b), (c |~#~| d))
```

# Java

```java
// generic

User user = mapper.readValue(json, User.class);
```

# Kotlin

```
// generic

fun <T, P> templateTest(value : T) : P {
  if ( value is P )  // Ошибка: фактический тип неизвестен
    return value
}

templateTest<String,String>("text")
```

# Kotlin

```
// generic

inline fun <T,reified P> templateTest(value : T) : P {
  if ( value is P )  // В этом случае тип известен и проверка возможна
    return value
}

templateTest<String,String>("text")
```

# Kotlin

```
inline fun <reifed T:Any> ObjectMapper.readValue(p: JsonParser): T =
                                readValue(p, object: TypeReference<T>() {})


val state: StateClass = mapper.readValue(json)
```

# Scala

```
// generic

TypeTag

WeakTypeTag

ClassTag
```

# Scala

```
// TypeTag

//def processClass[T: TypeTag](json: Map[String, T])

def processClass[T](json: Map[String, T])(implicit tag: TypeTag[T]) = {
    println(tag.tpe)
}

val clazz: Map[String, List[Int]] = Map("foo" -> List(1, 2, 3))

processClass(clazz)

// List[Int]
```

# Scala

```scala
// TypeTag

def processClass[T](json: Map[String, T])(implicit tag: TypeTag[T]) = {
    println(tag.tpe)
}

def foo[T](t: T) = processClass(t) // compilation error

val clazz: Map[String, List[Int]] = Map("foo" -> List(1, 2, 3))

foo(clazz)
```

# Scala

```
// WeakTypeTag

def processClass[T](json: Map[String, T])(implicit tag: WeakTypeTag[T]) = {
    println(tag.tpe)
}

def foo[T](t: T) = processClass(t)

val clazz: Map[String, List[Int]] = Map("foo" -> List(1, 2, 3))

foo(clazz)

// T
```

# Scala

```
// ClassTag

def processClass[T](t: T)(implicit tag: ClassTag[T]) = {
    println(tag.tpe)
}

val clazz: List[Int] = List(1, 2, 3)

processClass(clazz)

// List
```

# Метапрограммирование

- времени компиляции
- времени исполнения

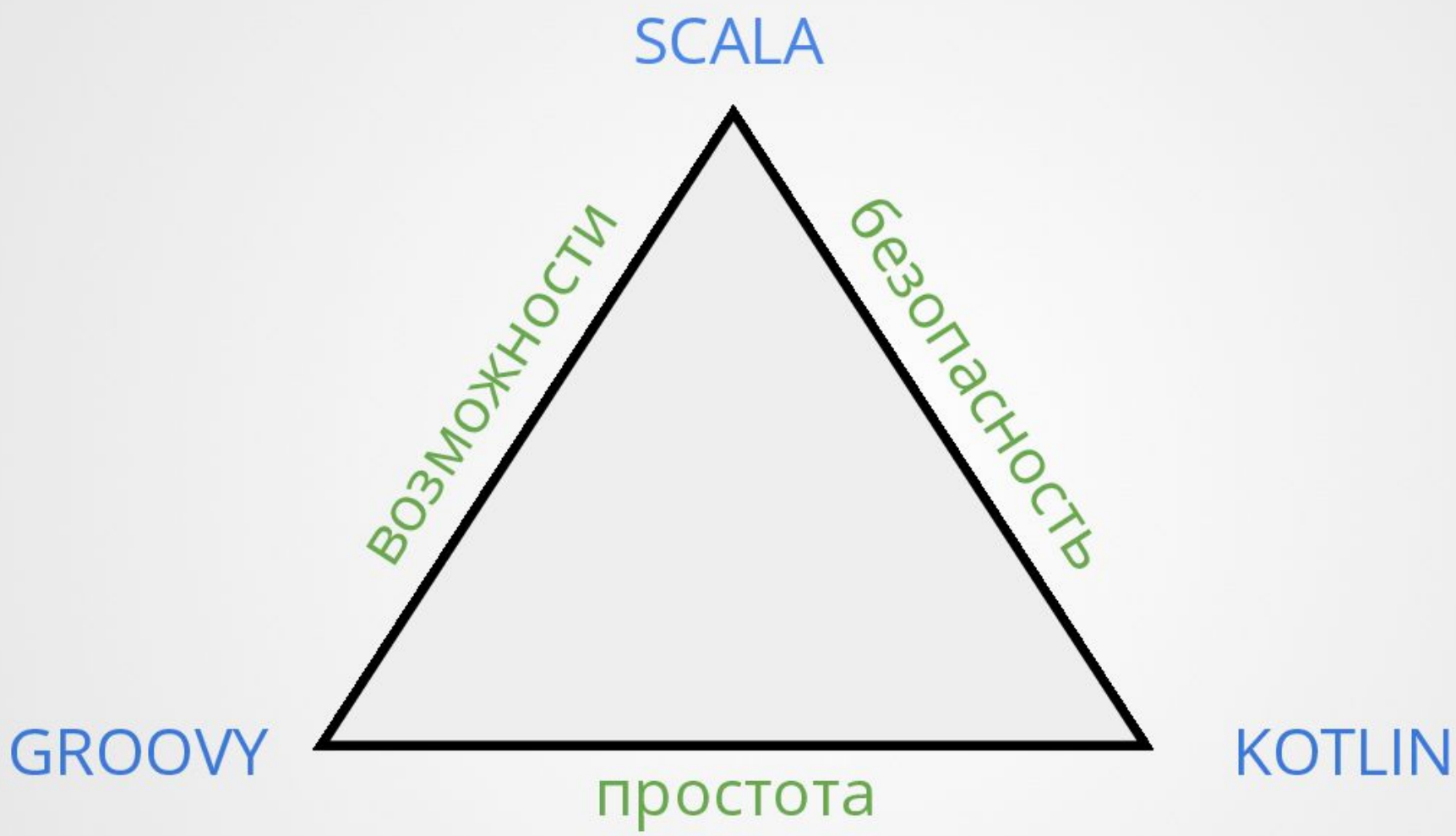
```groovy
// Property Missing

Integer.metaClass.propertyMissing = {String name ->
    if (name == 'seconds') {
        delegate * 1000
    }
}

t = 30.seconds
```

```groovy
// External Script

String script = new File("CoolLogic.dsl").text
new GroovyShell().evaluate(script)



def groovyCode ='sum=a+b; println "Sum is $sum"';

def binding = new Binding()
binding.a = 10
binding.a = 20

GroovyShell shell = new GroovyShell(binding);
shell.evaluate(groovyCode);
```

```groovy
// Method Missing


String.metaClass.methodMissing = { String name, args ->
    if (name.startsWith('doEvery')) {
        ...
    }
}

def account = "1234567890"

account.doEvery30Seconds()


Book.findByTitle("Groovy in Action")
```

```
// External Script

account check status every 30.minute
```

```
// AST Transformation

@Immutable

@Canonical

@PackageScope

@NotYetImplemented

@CompileStatic
```


SCALA
возможности
безопасность
GROOVY
KOTLIN
простота

# Немного о Clojure


ПРОШЛОЙ НОЧЬЮ, ЧИТАЯ КНИГУ ПРО LISP, Я ЗАДРЕМАЛ.
ЧТО?
И ВНЕЗАПНО ПОГРУЗИЛСЯ В ОСЛЕПИТЕЛЬНУЮ СИНЕВУ.


ТУТ ЖЕ, КАК И РАССКАЗЫВАЛИ, Я ОЩУТИЛ НЕВЕРОЯТНОЕ ПРОСВЕТЛЕНИЕ. Я УВИДЕЛ ЯСНУЮ СТРУКТУРУ LISP'ОВСКОГО КОДА, РАЗВЕРНУТУЮ ПЕРЕДО МНОЙ.
ГОСПОДИ
ОН ЖЕ ПОЛОН 'CAR'ОВ
ПАТТЕРНЫ И МЕТАПАТТЕРНЫ ПЛЯСАЛИ ВОКРУГ. СИНТАКСИС РАСТВОРИЛСЯ, И Я ПЛАВАЛ ЛИШЬ В ЧИСТОТЕ ПРОСЧИТАННЫХ КОНЦЕПЦИЙ И ПРОЗРАЧНЫХ ИДЕЙ.


ПОИСТИНЕ, ЭТО И БЫЛ ЯЗЫК, НА КОТОРОМ БОГИ РАЗРАБОТАЛИ ВСЕЛЕННУЮ.


ВООБЩЕ-ТО НЕТ.
КАК НЕТ?
ТО ЕСТЬ, ЯКОБЫ ДА. НО, ЕСЛИ ЧЕСТНО, БОЛЬШУЮ ЕЕ ЧАСТЬ МЫ НАСПЕХ СКОЛОТИЛИ НА PERL'Е.

```clojure
(def my-routes
  ["/notes" :index-handler])
```

```
GET /
GET /notes
GET /notes/:id
POST /notes
POST /notes/:id
```

```
(def my-routes
  ["" {"/" :home-page-handler
       "/notes"
       {:get  {"" :index-handler}
        :post {"" :create-handler}
        ["/" :id] {:get  {"" :show-handler}
                   :post {"" :update-handler}}}})
```

```clojure
(defn doc-sign [state] state)
(defn doc-delete [state] state)
(defn doc-archive [state] state)

(def document-sm
  {:actions [doc-sign doc-delete doc-archive]
   :states  {"new"
             {:actions [doc-sign doc-delete doc-archive]
              :transitions [{:signed "signed"}
                            :deleted "deleted"
                            :archived "archived"]}
             "signed"
             {:actions [doc-archive]
              :transitions {:archived "archived"}}
             "deleted"
             {:actions []
              :transitions []}
             "waiting-for-drawer"
             {:actions []
              :transitions []}}
   :initial-state :new})
```

```
(select
  (fields [:name :role_name])
  (from :Users)
  (join-inner :Roles (== :Users.role_id :Roles.id))
  (where (= :Roles.name "admin"))
  (order :Users.name)
  (limit 100))


(fetch-one db
  (select
    (from :Users)
    (where (== :id 123))))


(fetch-all db
  (select (from :Users)))
```

```clojure
(defn fetch-all
  [db relation]
    (jdbc/query
     db
     (to-sql-params relation)
     :result-set-fn vec))

(defn to-sql-params
  [relation]
  (let [{s :sql p :args} (as-sql relation)]
    (vec (cons s p))))
```

```clojure
(def empty-select {})

(-> empty-select
  (fields [:name :role_name])
  (from :Users)
  (limit 100))
```

```clojure
(-> empty-select
  (fields [:name :role_name])
  (from :Users)
  (limit 100))



; Макрос "->"
(limit
  (from
  (fields
    empty-select
    [:name :role_name])
  :Users)
100)
```

# Макрос это просто

```
(defmacro name doc-string? attr-map? ([params*] body)+)

; doc-string? - описание макроса (документация)
; attr-map?   - список атрибутов
```

```clojure
(defmacro select
  [& body]
   `(-> empty-select ~@body))
```

# Вопросы ?

Контакты:

https://github.com/solverit

solverit@gmail.com