Anatomy of a Modern Play Application

Marconi Lanna

Originate

Recollection of best practices and lessons we learned
successfully delivering multiple Scala + Play projects

Originate

Layer by layer

- Build and configuration
- Model
  - Persistence
- Service
- View
- Unit testing
- Alternatives

Build and configuration

Making Scala safer

"Toward a Safer Scala" - Leif Wickland

Build and configuration

Making Scala safer

- Enable `scalac` warning and lint options

Build and configuration

Making Scala safer

```
scalacOptions ++= Seq(
  // Emit warning for usages of deprecated APIs
    "-deprecation"
  // Emit warning for usages of features that should be imported explicitly
  , "-feature"
  // Enable additional warnings where generated code depends on assumptions
  , "-unchecked"
  // Fail the compilation if there are any warnings
  , "-Xfatal-warnings"
  // Enable or disable specific warnings
  , "-Xlint:_"
)
```

Build and configuration

Making Scala safer

```
$ scalac -Xlint:help
```

Option	Description
`adapted-args`	Warn if an argument list is modified to match the receiver
`by-name-right-associative`	By-name parameter of right associative operator
`delayedinit-select`	Selecting member of `DelayedInit`
`doc-detached`	A ScalaDoc comment appears to be detached from its element
`inaccessible`	Warn about inaccessible types in method signatures
`infer-any`	Warn when a type argument is inferred to be `Any`
`missing-interpolator`	A string literal appears to be missing an interpolator id

Build and configuration

Making Scala safer

Option	Description
`nullary-override`	Warn when non-nullary `def f()` overrides nullary `def f`
`nullary-unit`	Warn when nullary methods return `Unit`
`option-implicit`	`Option.apply` used implicit view
`package-object-classes`	Class or object defined in package object
`poly-implicit-overload`	Parameterized overloaded implicit methods are not visible as view bounds
`private-shadow`	A private field (or class parameter) shadows a superclass field
`type-parameter-shadow`	A local type parameter shadows a type already in scope
`unsound-match`	Pattern match may not be typesafe

Build and configuration

Making Scala safer

```
scalacOptions ++= Seq(
  // Do not adapt an argument list to match the receiver
    "-Yno-adapted-args"
  // Warn when dead code is identified
  , "-Ywarn-dead-code"
  // Warn when local and private vals, vars, defs, and types are are unused
  , "-Ywarn-unused"
  // Warn when imports are unused
  , "-Ywarn-unused-import"
  // Warn when non-Unit expression results are unused
  , "-Ywarn-value-discard"
)
```

Build and configuration

Making Scala safer

-Yno-adapted-args

```
$ scala

scala> List(1, 2, 3).toSet()


res0: Boolean = false


$ scala -Yno-adapted-args

scala> List(1, 2, 3).toSet()
error: not enough arguments for method apply: (elem: AnyVal)Boolean in trait
Unspecified value parameter elem.
              List(1,2,3).toSet()
                                ^
```

Build and configuration

Making Scala safer

```
scalacOptions ++= Seq(
  // Specify character encoding used by source files
    "-encoding", "UTF-8"
  // Target platform for object files
  , "-target:jvm-1.8"
  // Turn on future language features
  , "-Xfuture"
  // Compile without importing scala.*, java.lang.*, or Predef
  , "-Yno-imports"
  // Compile without importing Predef
  , "-Yno-predef"
)
```

Build and configuration

Making Scala safer

- Enable `scalac` warning and lint options
- Leverage static analysis tools

Build and configuration

Making Scala safer

WartRemover

https://github.com/puffnfresh/wartremover

```
addSbtPlugin("org.brianmckenna" % "sbt-wartremover" % "0.11")
```

- Inferred `Any`, `Nothing`, `Product`, `Serializable`
- `Option#get`
- `List#head`, `List#tail`, etc. (throw exception if empty)
- `null`
- `return`
- `throw`
- `var`

Build and configuration

Making Scala safer

- Enable `scalac` warning and lint options
- Leverage static analysis tools
- Enforce coding standards

Build and configuration

Making Scala safer

Scalastyle

Used by Martin Odersky Coursera classes

http://scalastyle.org/

```
addSbtPlugin("org.scalastyle" %% "scalastyle-sbt-plugin" % "0.6.0")
```

Build and configuration

Making Scala safer

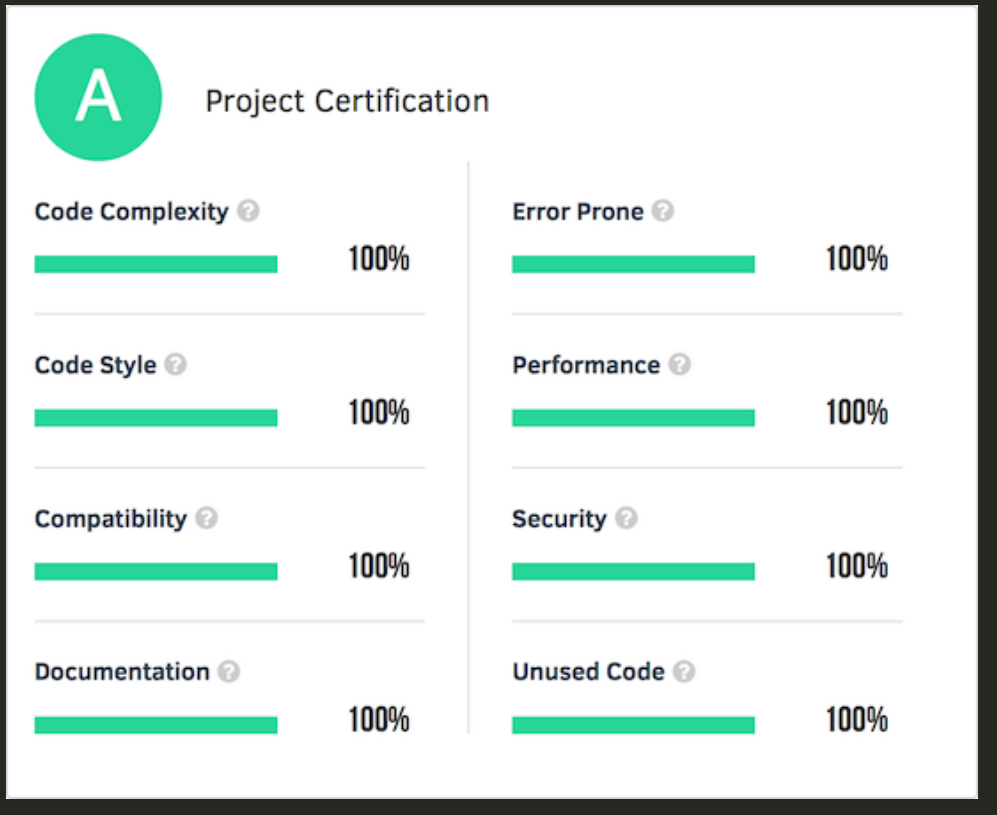
- Enable `scalac` warning and lint options
- Leverage static analysis tools
- Enforce coding standards
- Automated code review, continuous static analysis

Build and configuration

Making Scala safer

Codacy

http://codacy.com/


A
Project Certification
Code Complexity
100%
Error Prone
100%
Code Style
100%
Performance
100%
Compatibility
100%
Security
100%
Documentation
100%
Unused Code
100%

Jaime Jorge
@jaimefjorge

Follow

There's few things I enjoy more than seeing great code quality projects: codacy.com /public/marconi … Congrats @ScalaFacts!

Build and configuration

sbt options

Improved incremental compilation

sbt 0.13.2

```
incOptions := incOptions.value.withNameHashing(true)
```

- I've measured recompilation speed improvements from 25% to 80%
- The bigger the project, the better the improvement
- Your mileage may vary

Build and configuration

sbt options

Improved dependency management

sbt 0.13.7

```
updateOptions := updateOptions.value.withCachedResolution(true)
```

Build and configuration

sbt options

Dependency update

sbt-updates

Adds a `dependencyUpdates` sbt task that checks Maven repositories for dependency updates

https://github.com/rtimush/sbt-updates

```
addSbtPlugin("com.timushev.sbt" % "sbt-updates" % "0.1.8")
```

Build and configuration

Leveraging Typesafe Config HOCON

Typesafe Config is awesome. Really awesome.

https://github.com/typesafehub/config

Build and configuration

Leveraging Typesafe Config HOCON

Type-safe Typesafe Config config object

Fail early, fail fast, fail often

```
current.configuration.getString("auth.db.passwrod")
```

Did you see the typo?!?

Build and configuration

Leveraging Typesafe Config HOCON

```
class Settings(config: Configuration) {
  // Throw an exception at application initialization
  // if a key is missing, as recommended by
  // https://github.com/typesafehub/config#schemas-and-validation
  private def string(key: String): String = config.getString(key).get
  private def string(key: String, default: String): String =
      config.getString(key) getOrElse default
  // int, double, boolean...

  object auth {
    object db {
      val login = string("auth.db.login")
      val password = string("auth.db.password")
    }
  }

  object mail {
    val server = string("mail.server", "localhost")
  }

  val optional = config.getString("optional")
}
```

Build and configuration

Leveraging Typesafe Config HOCON

```
current.configuration.getString("auth.db.password")

settings.auth.db.password

settings.mail.server

settings.optional foreach { ... }
```

Build and configuration

Leveraging Typesafe Config HOCON

```
class Resource(resource: String) {
  private val config = ConfigFactory.parseFile(new File(resource))
      .resolve(ConfigResolveOptions.noSystem)

  def has(key: String): Boolean = cfg.hasPath(key)

  def apply(key: String): Option[String] =
    if (has(key)) Option(config.getString(key)) else None
  // int, double, boolean...
}
```

Model

Where to put business logic?

Thin or fat model?

Martin Fowler

"Domain Model: An object model of the domain that incorporates both behavior and data."

http://www.martinfowler.com/bliki/AnemicDomainModel.html

"[...] there is hardly any behavior on these objects, making them little more than bags of getters and setters."

"The fundamental horror of this anti-pattern is that it's so contrary to the basic idea of object-oriented design; which is to combine data and process together. The anemic domain model is really just a procedural style design."

Model

On the other hand...

God objects

Violate SOLID principles

Difficult to reason about

Difficult to debug

Difficult to unit test

Difficult to mock

Model

So, between a thin model or a fat model, why not have both?

In Scala, you can have your cake and eat it, too.

Model

Fat model (god object)

```
package model

case class User(email: Email, password: Password) {
  def authenticate(credentials: Credentials): Boolean = {
    email == credentials.email && password == credentials.password
  }
}


user.authenticate(credentials)
```

Model

Thin model, fat service

```
package model
case class User(email: Email, password: Password)

package service
class UserService {
  def authenticate(user: User, credentials: Credentials): Boolean = {
    user.email == credentials.email &&
      user.password == credentials.password
  }
}


userService.authenticate(user, credentials)
```

A little verbose, less elegant

Model

```
package model

case class User(email: Email, password: Password) extends UserAuth

trait UserAuth {
  self: User =>

  def authenticate(credentials: Credentials): Boolean = {
    email == credentials.email && password == credentials.password
  }
}


user.authenticate(credentials)
```

Same syntax as fat model for method signature, body, and invocation

Single responsibility principle: use different traits and keep them small and easy to reason about.

Easy to mock:

```
val mockUser = new User(email, password) with UserAuthMock
```

Model

Avoid "primitive" types (`Int`, `String`, `Date`, etc.)

Tiny types

Dick Wall: "Type Early, Type Often"

http://typesafe.com/blog/type-early-type-often

```
case class Person(first: String, last: String,
  address: String, age: Int, ssn: String)


case class Person(first: FirstName, last: LastName,
  address: Address, age: Age, ssn: SSN)


case class FirstName(name: String)


case class Age(age: Int) extends AnyVal
```

Model

require

```
case class Person(name: String, age: Int) {
  require(name.trim.nonEmpty)
  require(age >= 0)
  require(age <= 130)
}
```

Model

Email validator

```
import org.apache.commons.validator.routines.EmailValidator

case class Email(val email: String) {
  require(EmailValidator.getInstance.isValid(email))
}
```

Persistence

1. Avoid Anorm
2. Avoid Anorm
3. Seriously, avoid Anorm

Persistence

"Using JDBC is a **pain** [...] using the JDBC API directly is **tedious** [...]
We provide a **simpler** API for JDBC."

Linus Torvalds: "I see Subversion as being the most pointless project ever. Its slogan was 'CVS done right'. There is no way to do CVS right."

There is no way to do JDBC right!

Anorm: a **simpler** way to **pain** and **tedium**.

Persistence

Play's Computer Database sample app

To add a single field to a domain model, you have to update your code in 11 places.

The Anorm documentation page spends 11 paragraphs describing why type safety is not important.

Really?

Anorm: the worse case of throwing out the baby with the bath water.

Not type-safe, not DRY, too verbose, full of boilerplate.

Dude, seriously, avoid Anorm

Persistence

When implementing model persistence, wrap your API with `Future`

```
def get(id: Id[User]): User

def get(id: Id[User]): Future[User]
```

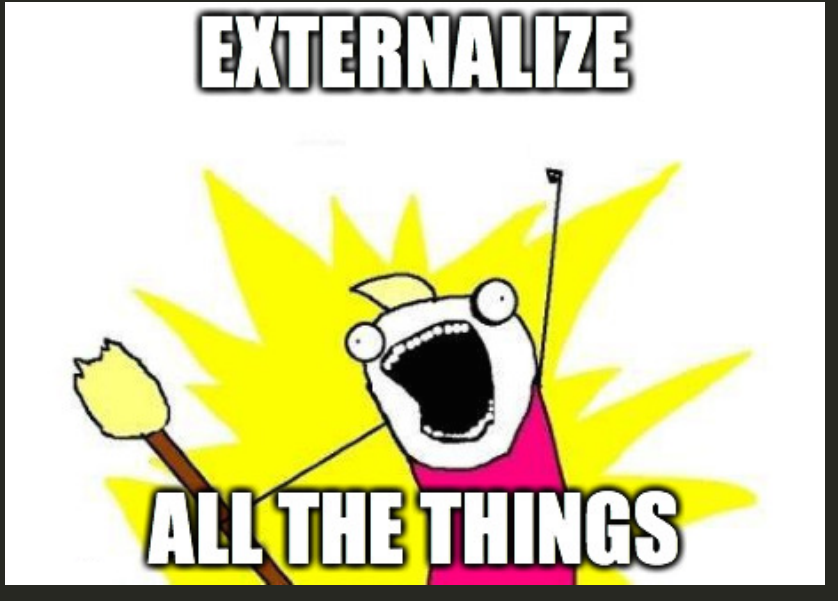
Prepare yourself for non-blocking IO

`Future.successful`

`scala.concurrent.blocking`

Persistence

No in-line SQL


EXTERNALIZE
ALL THE THINGS

Persistence

No in-line SQL

1. Keep code minimal
2. Change without recompiling
3. Give it to third parties (DBA)
4. Different sets of queries for different databases

Persistence

How?

Typesafe Config!

```
class Resource(resource: String) {
  def apply(key: String): Option[String] = ...
}


class Query(table: String) {
  private val query = Resource("conf/query/" + table + ".sql")

  def apply(q: String) = query(q).get
}


val userQuery = Query("user")

userQuery("findById")
```

Persistence

user.sql

```
include "common.sql.conf"

insert: """insert into user ..."""

update: """update user set ..."""

delete: """delete from user where id = {id}"""

select: """select * from user"""

findById: ${select}""" where id = {id}"""
```

Persistence

common.sql.conf

```
unicode: "collate utf8mb4_unicode_ci"
```

Persistence

MySQL

```
create database sample
  charset utf8mb4
  collate utf8mb4_bin;
```

View

Templates

Consider other template options

Twirl is not terrible.

Indeed, it has some pretty cool, unique features, like type safety.

Problem: embed full Scala code

Logic in templates

Compilation time

Syntax: sometimes things that look like they should work, don't.

Scalate: Scala Template Engine

http://scalate.github.io/scalate/

Client-side rendering

View

Custom tags

Easy to define powerful custom tags

```
<link rel="stylesheet" media="screen"
  href="@routes.Assets.at("stylesheets/main.css")">

<link rel="shortcut icon" type="image/png"
  href="@routes.Assets.at("images/favicon.png")">

<script src="@routes.Assets.at("javascripts/jquery-1.7.1.min.js")"
  type="text/javascript"></script>
```

View

Custom tags

```
sealed trait asset {
  protected def folder: String
  protected def ext = "." + folder
  def url(resource: String) =
    controllers.routes.Assets.at(s"$folder/$resource$ext").url
}

object css extends asset {
  protected def folder = "css"
  def apply(resource: String) =
    Html(s"""<link href="${url(resource)}" rel="stylesheet">""")
}

object js extends asset {
  protected def folder = "js"
  def apply(resource: String) =
    Html(s"""<script src="${url(resource)}"></script>""")
}
```

View

Custom tags

```
<link rel="stylesheet" media="screen"
  href="@routes.Assets.at("stylesheets/main.css")">

<link rel="shortcut icon" type="image/png"
  href="@routes.Assets.at("images/favicon.png")">

<script src="@routes.Assets.at("javascripts/jquery-1.7.1.min.js")"
  type="text/javascript"></script>
```

```
@css("bootstrap-3.3.4")
@js("jquery-2.1.3")
@img("logo.png")
```

Go crazy!

View

Advanced custom tags

```
@if (user.isAuthorized) {
  Welcome, @user.name!
} else {
  Nothing to see here...
}
```

Very tempting to litter the templates with business logic

```
@if (user.isAuthorized && user.isPremiumSubscriber) { ...
```

View

Advanced custom tags

```
// Extending NodeSeq is a little hack to prevent HTML double-escape
class IfTag(condition: Boolean, content: => Html)
  extends scala.xml.NodeSeq {

  def theSeq = Nil // just ignore, required by NodeSeq

  override def toString = if (condition) content.toString else ""

  def orElse(failed: => Html) = if (condition) content else failed
}


def isAuthorized(user: User)(body: => Html) =
  new IfTag(user.isAuthorized, body)


@isAuthorized(user){
  Welcome, @user.name!
}.orElse{
  Nothing to see here...
}
```

View

Asset pipeline

sbt-web

grunt/yeoman

gulp

Play-Yeoman

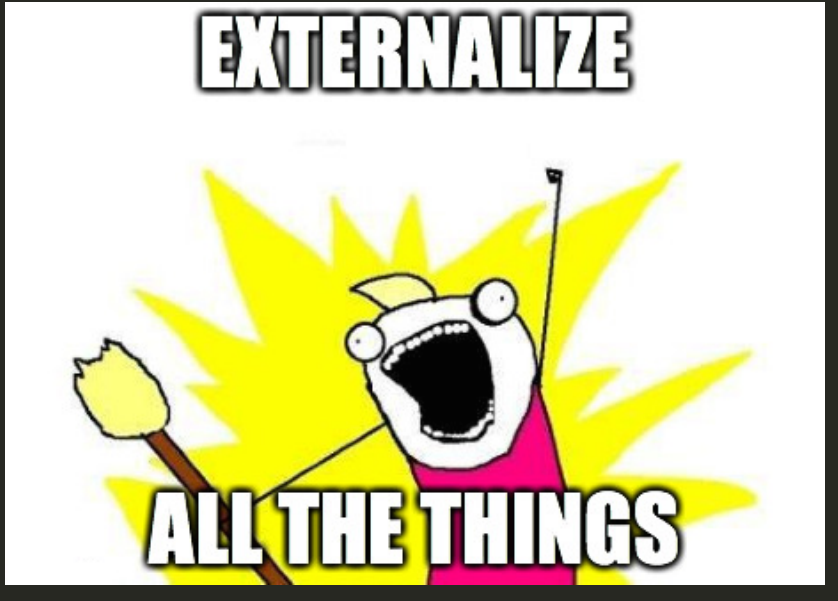
https://github.com/tuplejump/play-yeoman

View

Internationalization

It is easier to do it from the start.

Useful even if your application is targeted at only one country and one language.


EXTERNALIZE
ALL THE THINGS

View

Internationalization

1. Keep code minimal
2. Change without recompiling
3. Give it to third parties
4. Solve minor regional differences

View

Internationalization

Localization files are written using the Typesafe Config HOCON format.

Messages are rendered by ICU4J (International Components for Unicode)

Headed by IBM and used by Adobe, Amazon, Apache, Apple, Eclipse, Google, Intel, Mozilla...

Compared to Java `MessageFormat`, ICU4J supports named arguments, enhanced genders and plurals, cardinal (one, two, five thousand) and ordinal (1st, 2nd, 3rd) numbers, user-friendly apostrophe quoting syntax, full Unicode support, and much more.

Drop-in replacement

http://site.icu-project.org/

```
"com.ibm.icu" % "icu4j" % "54.1.1"
```

View

Internationalization

Solve minor regional differences

```
# messages.en-CA.conf

include "messages.en.conf"

state="province"
zipCode="postal code"
zipFormat="[a-zA-Z]\d[a-zA-Z] ?\d[a-zA-Z]\d"
```

View

HTML input sanitization

jsoup

Parses HTML, not regex-based

Converts relative links to absolute

Filter tags and attributes

Custom whitelists

http://jsoup.org/

```
"org.jsoup" % "jsoup" % "1.8.1"
```

Service

`Future`-based service architecture

"Your Server as a Function" by Marius Eriksen, Twitter

http://monkey.org/~marius/funsrv.pdf

```
type Service[Req, Res] = Req => Future[Res]
```

Service

Do not throw exceptions

Exceptions are not functional!

Functions that throw exceptions are not total functions

They do not return a value for all possible inputs

Use `Try` instead

`Try` does for exceptions what `Option` does for `null`

Unit testing

Prefer ScalaTest

Mocking: ScalaMock, Mockito

Code coverage: Scoverage

Dependency Injection

Avoid the Cake Pattern and the Bakery of Doom

Implicit parameters, MacWire

Alternatives

Not a criticism, but Play may not be always the best solution for all projects

Spray / Akka HTTP

Scalatra

References

- github.com/marconilanna/ScalaDaysSanFrancisco2015
- blog.originate.com

Thank you!

Questions?

Please share your experiences.