# scalaxb

eugene yokota

new york scala enthusiasts 2010-11-29

lately XML gets no ❤️

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:osgi="http://activemq.apache.org/camel/schema/osgi"
       xmlns:osgix="http://www.springframework.org/schema/osgi-compendium">

  <osgi:camelContext xmlns="http://activemq.apache.org/camel/schema/spring">
    <route>
      <from uri="timer://myTimer?fixedRate=true&amp;period=2000"/>
      <bean ref="myTransform" method="transform"/>
      <to uri="log:ExampleRouter"/>
    </route>
  </osgi:camelContext>

  <bean id="myTransform"
class="org.apache.servicemix.examples.camel.MyTransform">
    <property name="prefix" value="${prefix}"/>
  </bean>

  <osgix:property-placeholder persistent-id="org.apache.servicemix.examples">
    <osgix:default-properties>
      <prop key="prefix">MyTransform</prop>
    </osgix:default-properties>
  </osgix:property-placeholder>

</beans>
```

maybe it's because of all the <>

# XML is a metalanguage

# XML is a metalanguage

- a language that is a template for another language.

Schemas define the grammar of XML-based languages

# Schemas define the grammar of XML-based languages

- DTD is obsolete. everything is string.
- W3C XML Schema Definition Language (XSD) is the de facto standard. typed.
- Relax NG is easier to use. typed.

scalaxb is
an XML data binding tool
for Scala

# what’s data binding?

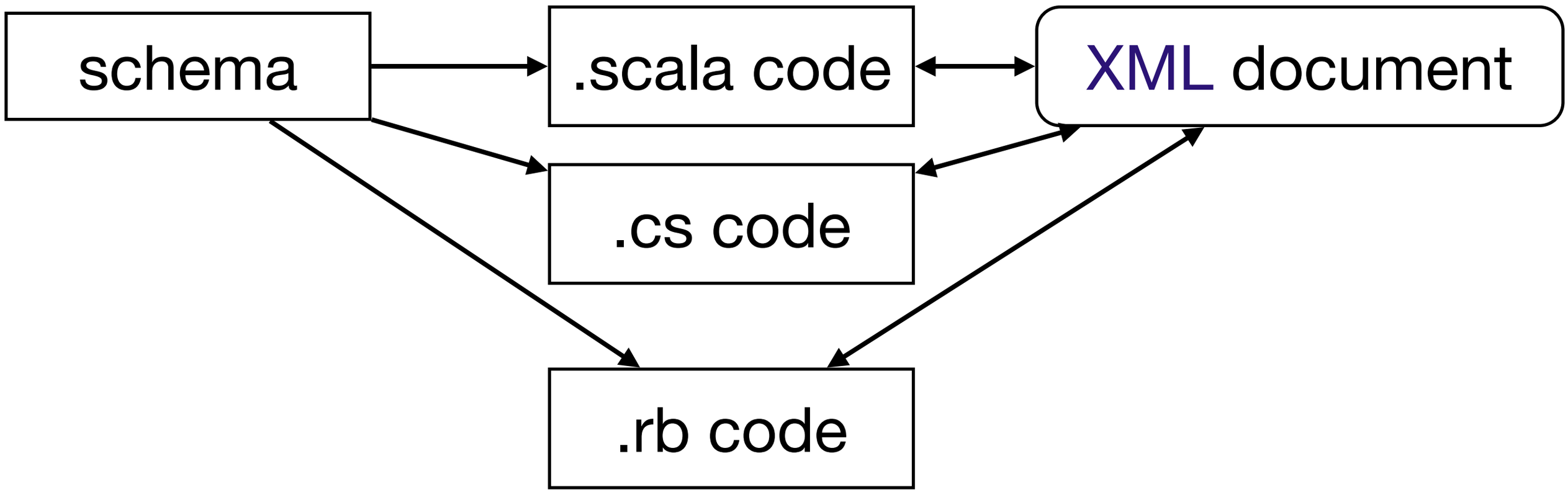
# XML serialization:

- I have bunch of objects.
  let's turn them into XML.

# XML data binding:

- I have bunch of XML documents.
  let's turn them into objects.

# XML data binding = cross-platform data exchange




profit!

scalaxb is
a W3C XML Schema
compiler for Scala

# XML document ≈ DOM

XML document ≈ DOM

DOM ≈ case class

XML document ≈ DOM

DOM ≈ case class

case class ≈ ❤️

# an example .rnc schema

```
default namespace = "http://www.example.com/address"

start =
  element addressBook {
    element card { cardContent }*
  }

cardContent =
  element name { text },
  element email { text },
  element zipCode { xsd:nonNegativeInteger }?
```

human-written schema

# $ trang demo.rnc demo.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified"
  targetNamespace="http://www.example.com/address"
  xmlns:address="http://www.example.com/address">
  <xs:element name="addressBook">
    <xs:complexType>
      <xs:element>
        <xs:element minOccurs="0" maxOccurs="unbounded" ref="address:card"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="card" type="address:cardContent"/>
  <xs:complexType name="cardContent">
    <xs:sequence>
      <xs:element ref="address:name"/>
      <xs:element ref="address:email"/>
      <xs:element minOccurs="0" ref="address:zipCode"/>
    </xs:sequence>
  </xs:complexType>
  <xs:element name="name" type="xs:string"/>
  <xs:element name="email" type="xs:string"/>
  <xs:element name="zipCode" type="xs:nonNegativeInteger"/>
</xs:schema>
```

# $ scalaxb demo.xsd -p address

## case class

```
// Generated by <a href="http://scalaxb.org/">scalaxb</a>.
package address

case class AddressBook(card: address.CardContent*)

case class CardContent(name: String,
  email: String,
  zipCode: Option[Int])
```

# $ scalaxb demo.xsd -p address

## case class

```
// Generated by <a href="http://scalaxb.org/">scalaxb</a>.
package address

case class AddressBook(card: address.CardContent*)

case class CardContent(name: String,
  email: String,
  zipCode: Option[Int])
```

- no XML nonsense here

```
// Generated by <a href="http://scalaxb.org/">scalaxb</a>.
package address

trait XMLProtocol extends scalaxb.XMLStandardTypes {
  implicit lazy val AddressAddressBookFormat: scalaxb.XMLFormat[address.AddressBook] =
    buildAddressAddressBookFormat
  def buildAddressAddressBookFormat: scalaxb.XMLFormat[address.AddressBook]

  implicit lazy val AddressCardContentFormat: scalaxb.XMLFormat[address.CardContent] =
    buildAddressCardContentFormat
  def buildAddressCardContentFormat: scalaxb.XMLFormat[address.CardContent]
}
```

- XMLProtocol defines implicit values of XMLFormat[AddressBook] and XMLFormat[CardContent]

```
// Generated by <a href="http://scalaxb.org/">scalaxb</a>.
package address

trait XMLProtocol extends scalaxb.XMLStandardTypes {
  implicit lazy val AddressAddressBookFormat: scalaxb.XMLFormat[address.AddressBook] =
    buildAddressAddressBookFormat
  def buildAddressAddressBookFormat: scalaxb.XMLFormat[address.AddressBook]

  implicit lazy val AddressCardContentFormat: scalaxb.XMLFormat[address.CardContent] =
    buildAddressCardContentFormat
  def buildAddressCardContentFormat: scalaxb.XMLFormat[address.CardContent]
}
```

# XMLFormat[A] is a typeclass contract

```
trait XMLFormat[A] extends CanWriteXML[A] with CanReadXML[A]

trait CanReadXML[A] {
  def reads(seq: scala.xml.NodeSeq): Either[String, A]
}

trait CanWriteXML[A] {
  def writes(obj: A, namespace: Option[String], elementLabel: Option[String],
      scope: scala.xml.NamespaceBinding, typeAttribute: Boolean): scala.xml.NodeSeq
}
```

# what's a typeclass again?

- a set of functions
- that can be called over arbitrary data type (including Int) if a typeclass instance is implemented
- typeclasses can be implemented in Scala using implicit parameters

Scala Implicits: Type Classes Here I come [Ghosh]

# object Scalaxb defines typeclass API

```
package scalaxb

object Scalaxb {
  def fromXML[A](seq: scala.xml.NodeSeq)(implicit format: XMLFormat[A]): A =
    format.reads(seq) match {
      case Right(a) => a
      case Left(a) => error(a)
    }

  def toXML[A](obj: A, namespace: Option[String], elementLabel: Option[String],
      scope: scala.xml.NamespaceBinding, typeAttribute: Boolean)
      (implicit format: CanWriteXML[A]): scala.xml.NodeSeq =
    format.writes(obj, namespace, elementLabel, scope, typeAttribute)
}
```

# XMLFormat[A] is the typeclass contract

```
trait XMLFormat[A] extends CanWriteXML[A] with CanReadXML[A]

trait CanReadXML[A] {
  def reads(seq: scala.xml.NodeSeq): Either[String, A]
}

trait CanWriteXML[A] {
  def writes(obj: A, namespace: Option[String], elementLabel: Option[String],
      scope: scala.xml.NamespaceBinding, typeAttribute: Boolean): scala.xml.NodeSeq
}
```

# typeclass instance for AddressBook

```
trait DefaultXMLProtocol extends XMLProtocol {
  import scalaxb.Scalaxb._

  override def buildAddressAddressBookFormat = new DefaultAddressAddressBookFormat {}
  trait DefaultAddressAddressBookFormat extends
      scalaxb.ElemNameParser[address.AddressBook] {
    val targetNamespace: Option[String] = Some("http://www.example.com/address")

    def parser(node: scala.xml.Node): Parser[address.AddressBook] =
      rep(scalaxb.ElemName(targetNamespace, "card")) ^^
      { case p1 =>
      address.AddressBook(p1.toSeq map { fromXML[address.CardContent](_) }: _*) }

    def writesChildNodes(__obj: address.AddressBook,
      __scope: scala.xml.NamespaceBinding): Seq[scala.xml.Node] =
      (__obj.card flatMap {
        toXML[address.CardContent](_, targetNamespace, Some("card"), __scope, false) })

  }

...
```

- ElemNameParser[A] implements XMLFormat[A] by using parser combinator.

# typeclass instance for CardContent

```
...

  override def buildAddressCardContentFormat = new DefaultAddressCardContentFormat {}
  trait DefaultAddressCardContentFormat extends
      scalaxb.ElemNameParser[address.CardContent] {
    val targetNamespace: Option[String] = Some("http://www.example.com/address")

    override def typeName: Option[String] = Some("cardContent")

    def parser(node: scala.xml.Node): Parser[address.CardContent] =
      (scalaxb.ElemName(targetNamespace, "name")) ~
      (scalaxb.ElemName(targetNamespace, "email")) ~
      opt(scalaxb.ElemName(targetNamespace, "zipCode")) ^^
      { case p1 ~ p2 ~ p3 =>
      address.CardContent(fromXML[String](p1),
        fromXML[String](p2),
        p3.headOption map { fromXML[Int](_) }) }

    def writesChildNodes(__obj: address.CardContent,
      __scope: scala.xml.NamespaceBinding): Seq[scala.xml.Node] =
      Seq.concat(
        toXML[String](__obj.name, targetNamespace, Some("name"), __scope, false),
        toXML[String](__obj.email, targetNamespace, Some("email"), __scope, false),
        __obj.zipCode map {
         toXML[Int](_, targetNamespace, Some("zipCode"), __scope, false)
        } getOrElse {Nil})
  }
}
```

# example usage

```
import scalaxb._
import Scalaxb._
import address._
import DefaultXMLProtocol._

object Usage extends Application {
  val addressBook = fromXML[AddressBook](<addressBook
        xmlns="http://www.example.com/address">
      <card><name>foo</name><email>foo@example.com</email></card>
      <card><name>bar</name><email>bar@example.com</email></card>
    </addressBook>)

  val modified = AddressBook(addressBook.card map { card =>
    card.copy(name = card.name + "2")}: _*)
  println(toXML[AddressBook](modified, Some("http://www.example.com/address"),
    Some("addressBook"), defaultScope))
}
```

# human-written code

# example usage

```
import scalaxb._
import Scalaxb._
import address._
import DefaultXMLProtocol._

object Usage extends Application {
  val addressBook = fromXML[AddressBook](<addressBook
        xmlns="http://www.example.com/address">
      <card><name>foo</name><email>foo@example.com</email></card>
      <card><name>bar</name><email>bar@example.com</email></card>
    </addressBook>) (DefaultXMLProtocol.AddressAddressBookFormat)

  val modified = AddressBook(addressBook.card map { card =>
    card.copy(name = card.name + "2")}: _*)
  println(toXML[AddressBook](modified, Some("http://www.example.com/address"),
    Some("addressBook"), defaultScope) (DefaultXMLProtocol.AddressAddressBookFormat))
}
```

compiler injects implicit value from the enclosing lexical scope.

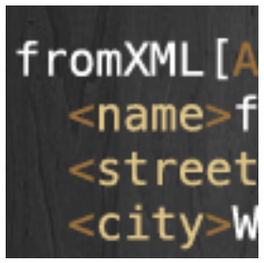
# example usage

```
import scalaxb._
import Scalaxb._
import address._
import DefaultXMLProtocol._

object Usage extends Application {
  val addressBook = fromXML[AddressBook](<addressBook
        xmlns="http://www.example.com/address">
      <card><name>foo</name><email>foo@example.com</email></card>
      <card><name>bar</name><email>bar@example.com</email></card>
    </addressBook>)

  val modified = AddressBook(addressBook.card map { card =>
    card.copy(name = card.name + "2")}: _*)
  println(toXML[AddressBook](modified, Some("http://www.example.com/address"),
    Some("addressBook"), defaultScope))
}
```

```
$ scalac demo.scala xmlprotocol.scala scalaxb.scala usage.scala
$ scala Usage
<addressBook xmlns="http://www.example.com/address" xmlns:address="http://
www.example.com/address" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"><card><name>foo2</name><email>foo@example.com</email></
card><card><name>bar2</name><email>bar@example.com</email></card></addressBook>
```

http://scalaxb.org

@eed3si9n




@scalaxb

eugene yokota <eed3si9n at gmail>