# Manipulação de XML em Java

Prof. Dr. Alexandre L'Erario


# Conjunto generoso de bibliotecas

- Existem muitas soluções
  - JAXB 2 e JAXB 3 (Jakarta XML bind API) – JAXB 4 (EE10)
  - DOM
  - Sax
  - XMLBeans(apache)
  - Xstream – lib java
  - JacksonXML – lib java

# JaxB

- Jakarta XML bind API
- Versão incorporada e retomada do JaxB
- Melhoras de bugs e performance
- Migração de pacotes (para jakarta.)
- Manipulação fácil de XML
- Incorporado na especificação EE9 (nov/2020)
  - https://eclipse-ee4j.github.io/jaxb-ri/
- Agora com o jxc novamente! ☺

# JaxB

- Architecture for XML Binding (JAXB)
  - Acessa e atualiza representação Java
  - Unmarshal:  XML → Java
  - Marshal:  Java → XML

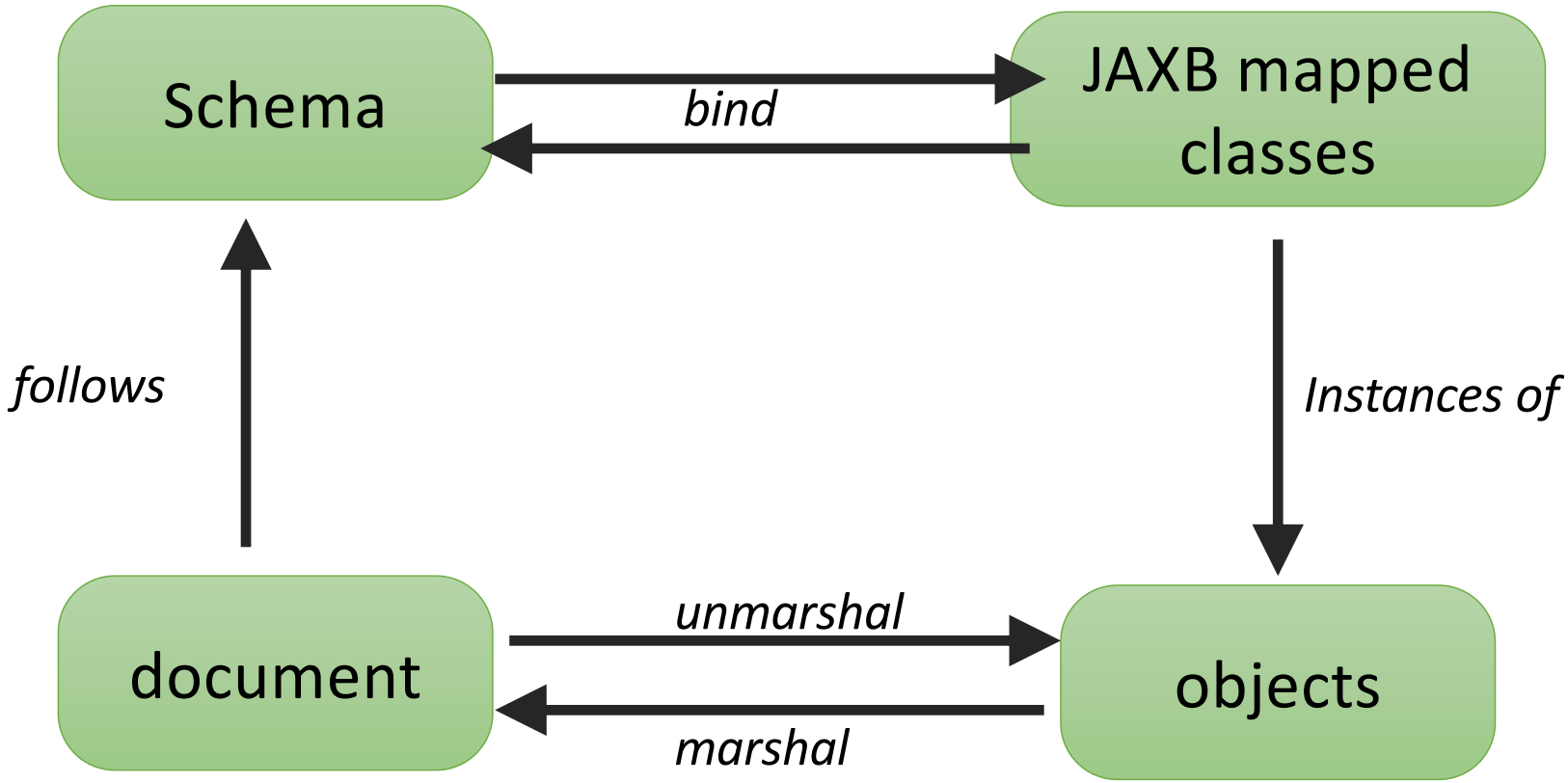
- Vantagens:
  - JAXB modelo eficiente e padronizado de mapear XML em código Java
  - Mais produtivo e menos código
  - Não precisa de experiência em XML

https://javaee.github.io/jaxb-v2/

# JaxB – Implementação

- Trata documentos com java annotations;
- Utiliza padrões de projeto na construção de sua arquitetura (factory, builder)
- Pode analisar pedaços de documentos XML
  - document by chunk
  - Pode ser utilizado para processar Streams
- Navega no documento como navega em objetos Java

# JaxB – Binding Process



# JaxB – Performance

- A classe JAXBContext (entry point API) é **thread-safe**, mas:
  - Marshaller, Unmarshaller, and Validator não;
  - Thread-save – semáfaro java a partir do 1.2

- Em aplicações multi-thread recomenda-se criar um singleton para a classe JAXBContext
  - JaxB pode ser extensa, mas foi projetada para vazão

# Transformando XSD em classes Java

- https://eclipse-ee4j.github.io/jaxb-ri/
- É possível fazer a transformação XSD em .java com o XJC
- XJC é um compilador capaz de manipular o formato XML

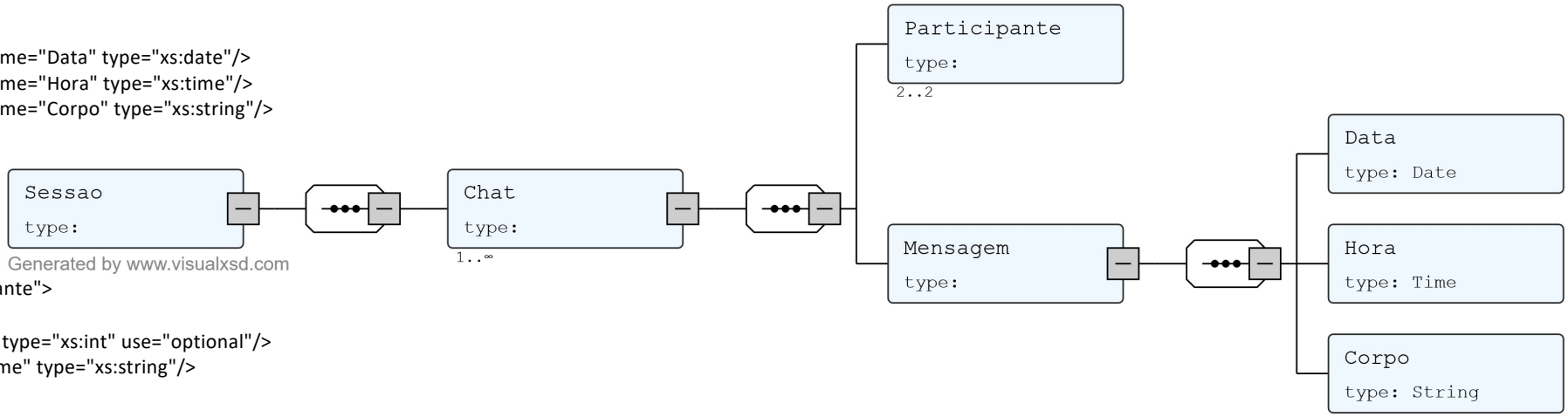
**#xjc.sh chat.xsd -p xml.pkg -encoding UTF-8**

Onde: -p <nome_do_pacote>

https://eclipse-ee4j.github.io/jaxb-ri/3.0.0/docs/ch03.html#compiling-xml-schema

# Analisando um XSD – Exemplo aplicado

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <xs:element name="Sessao">
    <xs:complexType>
      <xs:sequence minOccurs="1">
        <xs:element ref="Chat" maxOccurs="unbounded" minOccurs="1"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="Chat">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Participante" maxOccurs="2" minOccurs="2"/>
        <xs:element name="Mensagem">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Data" type="xs:date"/>
              <xs:element name="Hora" type="xs:time"/>
              <xs:element name="Corpo" type="xs:string"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="Participante">
    <xs:complexType>
      <xs:attribute name="id" type="xs:int" use="optional"/>
      <xs:attribute name="nome" type="xs:string"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```



# Resultado após execução do XJC


```
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "", propOrder = {
   "chat"
})
@XmlRootElement(name = "Sessao")
public class Sessao {
   @XmlElement(name = "Chat", required = true)
   protected List<Chat> chat;
public List<Chat> getChat() {
      if (chat == null) {
         chat = new ArrayList<Chat>();
      }
      return this.chat;
   }
}
```

O compilador também gera o Factory

# Pom.xml – Dependência de pacotes no Maven

```
<dependencies>
        <dependency>
...
        <dependency>
                <groupId>jakarta.xml.bind</groupId>
                <artifactId>jakarta.xml.bind-api</artifactId>
                <version>3.0.1</version>
        </dependency>
        <dependency>
                <groupId>org.glassfish.jaxb</groupId>
                <artifactId>jaxb-runtime</artifactId>
                <version>3.0.2</version>
        </dependency>
</dependencies>
```

API

Implementação da API

# Criar objetos – Com classes POJO

```
Sessao ss = new Sessao();                          Chat ct = new Chat();
Participante P1  = new Participante();             Participante P2 = new Participante();
P1.setNome("Alexandre");                           P1.setId(1);
P2.setNome("Flavia");                              P2.setId(2);
Chat.Mensagem msg = new Chat.Mensagem();
msg.setCorpo("Oi como vai tudo bem?");
GregorianCalendar dt = new GregorianCalendar(); //precisa de try e catch
XMLGregorianCalendar xmlDate = DatatypeFactory.newInstance().newXMLGregorianCalendar(dt);
msg.setData(xmlDate);                              msg.setHora(xmlDate);
ct.setMensagem(msg);
ct.participante = new ArrayList<Participante>();
ct.participante.add(P1);                           ct.participante.add(P2);
ss.chat = new ArrayList<Chat>();
ss.chat.add(ct);
```

Toda manipulação foi executada sem nenhuma modificação nas classes criadas pelo XJC

# Exemplo de marshal

```
//Criar um objeto Sessao (ss) estilo POJO
try {
        JAXBContext context =
JAXBContext.newInstance(ObjectFactory.class);
        String txt = "";
        context.createMarshaller().marshal(ss, System.out);
        StringWriter sw = new StringWriter();
        context.createMarshaller().marshal(ss, sw);
        txt = sw.toString();
```

Classe criada pelo XJC

# Exemplo de unmarshal

```
try {
        JAXBContext context = JAXBContext.newInstance(ObjectFactory.class);
        Unmarshaller u = context.createUnmarshaller();
        StringReader reader = new StringReader(txt);
        Sessao sess = (Sessao) u.unmarshal(reader);

         //agora podemos tratar sess como um POJO
        for (Participante part : sess.chat.get(0).participante) {
           System.out.println("nome:" + part.getNome());
        }
```

# Java Annotations

- Gerar o schema ou XML a partir de classes POJO
- Existem muitas annotations:

  https://jakarta.ee/specifications/xml-binding/3.0/apidocs/jakarta.xml.bind/jakarta/xml/bind/annotation/package-summary.html
- Determinam a estrutura do XSD/XML
- Determinam se um atributo na classe é um atributo no XML, por exemplo

# Exemplo - Classes Disciplina e Aluno

```
@XmlRootElement(name = "disciplina")
public class Disciplina {
   private String nome;
   public Disciplina() {
   }
public Disciplina(String nome) {
      this.nome = nome;
   }
@XmlElement(name="nome")
   public String getNome() {
      return nome;
   }

public void setNome(String nome) {
      this.nome = nome;
   }

}
```

```
@XmlRootElement(name = "aluno")
public class Aluno implements Serializable{

   @XmlElementWrapper(name = "disciplinas", namespace = "")
   @XmlElement(name="disciplina", namespace = "")
   public ArrayList<Disciplina> getLdisciplina() {
      return ldisciplina;
   }

 @XmlAttribute
   public int getId() {
      return id;
   }


   @XmlAttribute
   public String getRa() {
      return ra;
   }
...
}
```

Será substituído por “disciplinas” no XML


*https://github.com/alerario/XmlOPS/tree/master/XmlOps/src/main/java/br/model*

# Exemplo – Classe Alunos e XSD gerado

```
@XmlRootElement(name = "lista_alunos")
public class Alunos implements java.io.Serializable{
   private java.util.ArrayList<Aluno> laluno;
public Alunos(ArrayList<Aluno> laluno) {
      this.laluno = laluno;
   }
 public Alunos() {
   }
   @XmlElement(name="aluno")
   public ArrayList<Aluno> getLaluno() {
      return laluno;
   }
   public void setLaluno(ArrayList<Aluno> laluno) {
      this.laluno = laluno;
   }
}
```

```
<?xml version="1.0" standalone="yes"?>
<xs:schema version="1.0" xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="aluno" type="aluno"/>
 <xs:element name="disciplina" type="disciplina"/>
 <xs:element name="lista_alunos" type="alunos"/>
<xs:complexType name="alunos">
  <xs:sequence>
   <xs:element ref="aluno" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
 </xs:complexType>
 <xs:complexType name="aluno">
  <xs:sequence>
   <xs:element name="disciplinas" minOccurs="0">
    <xs:complexType>
     <xs:sequence>
      <xs:element ref="disciplina" minOccurs="0" maxOccurs="unbounded"/>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
  </xs:sequence>
  <xs:attribute name="id" type="xs:int" use="required"/>
  <xs:attribute name="nome" type="xs:string"/>
  <xs:attribute name="ra" type="xs:string"/>
 </xs:complexType>

 <xs:complexType name="disciplina">
  <xs:sequence>
   <xs:element name="nome" type="xs:string" minOccurs="0"/>
  </xs:sequence>
 </xs:complexType>
</xs:schema>
```

# Exemplo de código para gerar o XSD

```
try {
        JAXBContext jaxbContext = JAXBContext.newInstance(br.model.Alunos.class);
        MySchemaOutputResolver sor;  = new MySchemaOutputResolver();
        jaxbContext.generateSchema(sor);
        System.out.println(sor.getSchema());
...
private static class MySchemaOutputResolver extends SchemaOutputResolver {
      public MySchemaOutputResolver() {   }
      private StringWriter stringWriter = new StringWriter();
      public Result createOutput(String namespaceURI, String suggestedFileName) throws IOException {
         StreamResult result = new StreamResult(stringWriter);
         result.setSystemId(suggestedFileName);
         return result;
      }
      public String getSchema() { return stringWriter.toString();
      }
   }
```

# Considerações finais

- Retomada do JaxB (XJC + Annotations)